Fast Computation of k-Runs, Parameterized Squares, and Other Generalised Squares

Yuto Nakashima
Department of Informatics, Kyushu University, Fukuoka, Japan

Jakub Radoszewski
Institute of Informatics, University of Warsaw, Poland

Tomasz Waleń
Institute of Informatics, University of Warsaw, Poland

Abstract

A k-mismatch square is a string of the form XY where X and Y are two equal-length strings that have at most k mismatches. Kolpakov and Kucherov [*Theor. Comput. Sci.*, 2003] defined two notions of k-mismatch repeats, called k-repetitions and k-runs, each representing a sequence of consecutive k-mismatch squares of equal length. They proposed algorithms for computing k-repetitions and k-runs working in $\mathcal{O}(nk \log k + \mathsf{output})$ time for a string of length n over an integer alphabet, where output is the number of the reported repeats. We show that $\mathsf{output} = \mathcal{O}(nk \log k)$, both in case of k-repetitions and k-runs, which implies that the complexity of their algorithms is actually $\mathcal{O}(nk \log k)$. We apply this result to computing parameterized squares.

A parameterized square is a string of the form XY such that X and Y parameterized-match, i.e., there exists a bijection f on the alphabet such that $f(X) = Y$. Two parameterized squares XY and $X'Y'$ are equivalent if they parameterized match. Recently Hamai et al. [SPIRE 2024] showed that a string of length n over an alphabet of size σ contains less than $n\sigma$ non-equivalent parameterized squares, improving an earlier bound by Kociumaka et al. [*Theor. Comput. Sci.*, 2016]. We apply our bound for k-mismatch repeats to propose an algorithm that reports all non-equivalent parameterized squares in $\mathcal{O}(n\sigma \log \sigma)$ time. We also show that the number of non-equivalent parameterized squares can be computed in $\mathcal{O}(n \log n)$ time. This last algorithm applies to squares under any substring compatible equivalence relation and also to counting squares that are distinct as strings. In particular, this improves upon the $\mathcal{O}(n\sigma)$-time algorithm of Gawrychowski et al. [CPM 2023] for counting order-preserving squares that are distinct as strings if $\sigma = \omega(\log n)$.

2012 ACM Subject Classification Theory of computation → Pattern matching

Keywords and phrases string algorithm, k-mismatch square, parameterized square, order-preserving square, maximum gapped repeat

Digital Object Identifier 10.4230/LIPIcs.ESA.2025.8

Related Version *Full Version*: `https://arxiv.org/abs/2509.02179`

Funding *Yuto Nakashima*: JSPS KAKENHI Grant Number JP25K00136.
Jakub Radoszewski: Supported by the Polish National Science Center, grant no. 2022/46/E/ST6/00463.

1 Introduction

A string of the form XX, for any string X, is called a *square* (or a *tandem repeat*). Squares are a classic notion in combinatorics on words (see, e.g., the early works of Thue [53] on avoidability of square substrings), text algorithms (starting from an algorithm for computing square substrings by Main and Lorentz [40]), and bioinformatics (see Gusfield's book [22]). A string of length n contains at most n distinct square substrings [8] and all of them can be computed in $\mathcal{O}(n)$ time [4, 9, 14, 23] or even $\mathcal{O}(n/\log_\sigma n)$ time [10] assuming that the string is over an integer alphabet of size σ. A maximal sequence of consecutive squares in a string is called a run (or a generalised run; cf. Section 2); see [33]. A string of length n contains at most n runs [3] and they can all be computed in $\mathcal{O}(n)$ time [3, 17]. Our work is devoted to efficient algorithms for computing known generalizations of squares and runs: k-mismatch squares represented as k-runs or k-repetitions, parameterized squares, and generalised squares that include parameterized squares, order-preserving squares, and Cartesian-tree squares.

We assume that positions in a string X are numbered from 1 to $|X|$, so that $X[i]$ is the ith character of X. For integers i, j such that $1 \le i \le j \le |X|$, by $X[i\,..\,j] = X[i\,..\,j+1)$ we denote the substring composed of characters $X[i], X[i+1], \ldots, X[j]$. We use similar notation for integer intervals: $[i\,..\,j] = [i\,..\,j+1) = \{i, i+1, \ldots, j\}$.

We say that a length-n string is over an integer alphabet if its letters belong to $[0\,..\,n^{\mathcal{O}(1)}]$. We use the word-RAM model of computation.

1.1 k-Mismatch Squares and k-Runs

For two equal-length strings X and Y, their Hamming distance is defined as $d_H(X,Y) = |\{i \in [1\,..\,|X|] : X[i] \ne Y[i]\}|$. A *$k$-mismatch square* (also known under the name of k-mismatch tandem repeat) is a string XY such that $|X| = |Y|$ and $d_H(X,Y) \le k$. Let T be a string of length n. A *k-run of period ℓ* in T (cf. [38]) is a maximal substring $T[a\,..\,b)$ such that $T[i\,..\,i+2\ell)$ is a k-mismatch square for every $i \in [a\,..\,b-2\ell]$; see Figure 1. Maximality means that the k-run is extended to the right and left as much as possible provided that the definition is still satisfied.

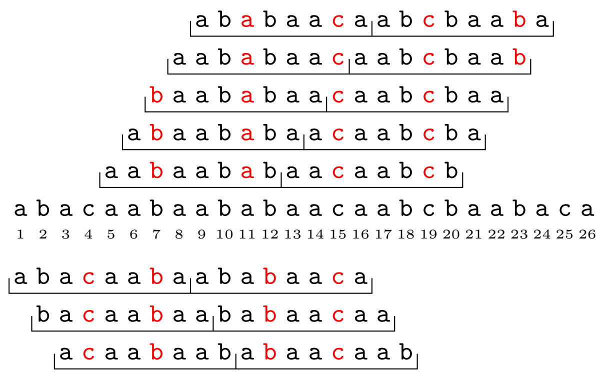

Figure 1 String $T[1\,..\,26]$ contains two 2-runs with period 8, $T[1\,..\,18]$ and $T[5\,..\,24]$. The two 2-runs represent three 2-mismatch squares (below) and five 2-mismatch squares (above), respectively; mismatches are shown in red. Strings $T[x\,..\,x+16)$ for $x \in \{4, 10, 11\}$ are not 2-mismatch squares.

Landau, Schmidt and Sokol [39] showed an algorithm for computing k-runs in a string of length n that works in $\mathcal{O}(nk\log(n/k))$ time; hence, the total number of k-runs reported is $\mathcal{O}(nk\log(n/k))$. Kolpakov and Kucherov [34, 35] showed that all k-runs (called there runs of k-mismatch tandem repeats) in a string of length n can be computed in $\mathcal{O}(nk\log k + \mathsf{output})$ time, where output is the number of k-runs. Many other algorithms for computing approximate tandem repeats under various metrics, in the context of computational biology and with the aid of statistical methods and heuristics, were proposed [6, 7, 15, 16, 28, 31, 36, 43, 45, 46, 47, 48, 49, 50, 51, 52, 55, 56]. In Section 2 we show the following theorem.

▶ **Theorem 1.** *A string of length n contains $\mathcal{O}(nk\log k)$ k-runs.*

Theorem 1 provides the first $\mathcal{O}(n)$ upper bound on the number of k-runs for $k = \mathcal{O}(1)$ and implies that the algorithm of Kolpakov and Kucherov computing k-runs actually works in $\mathcal{O}(nk\log k)$ time. Actually, we show a stronger condition that a string T of length n

contains $\mathcal{O}(nk \log k)$ *uniform* k-runs. Intuitively, a uniform k-run is a maximal sequence of consecutive k-mismatch squares of the same length in which the mismatches of all squares are at the same positions. (See Section 2 for a formal definition.) Our bound on the number of uniform k-runs implies the bound for k-runs as well as an $\mathcal{O}(nk \log k)$ upper bound on the number of k-repetitions as defined in [34, 35] (called there k-mismatch globally defined repetitions).

To prove Theorem 1, we explore a combinatorial relation between k-runs and maximum gapped repeats [32] and apply the optimal $\mathcal{O}(n\alpha)$ bound on the number of maximal α-gapped repeats in a length-n string [20, 27].

1.2 Parameterized Squares

For a string X, by $\mathsf{Alph}(X)$ we denote the set of characters of X. Two strings X, Y *parameterized match* if $|X| = |Y|$ and there is a bijection $f : \mathsf{Alph}(X) \mapsto \mathsf{Alph}(Y)$ such that $f(X) = Y$ (i.e., $|X| = |Y|$ and $f(X[i]) = Y[i]$ for all $i \in [1\mathinner{.\,.}|X|]$). A *parameterized square* (*p-square*, in short) is a string XY such that X parameterized matches Y (see Figure 2). Two p-squares XY and $X'Y'$ are called *equivalent* if they parameterized match.

Parameterized matching was introduced by Baker [2] motivated by applications in code refactoring and plagiarism detection. The notion of p-squares was introduced by Kociumaka, Radoszewski, Rytter, and Waleń [30] who showed that a string of length n over alphabet of size σ contains at most $2\sigma!n$ non-equivalent p-squares. They also considered avoidability of parameterized cubes. The bounds from [30] were recently improved by Hamai, Taketsugu, Nakashima, Inenaga, and Bannai [24]; they showed that a length-n string over alphabet of size σ contains less than σn non-equivalent p-squares. This bound automatically implies that such a string contains at most $\sigma! \cdot \sigma \cdot n$ p-squares that are distinct as strings (improving the $2(\sigma!)^2 n$ upper bound from [30]). We show that all non-equivalent p-squares can be computed efficiently; our approach also extends to p-squares that are distinct as strings. See Section 4.

▶ **Theorem 2.** *All non-equivalent p-square substrings in a string of length n over alphabet of size σ can be computed in $\mathcal{O}(n\sigma \log \sigma)$ time.*

All p-square substrings that are distinct as strings can be computed in $\mathcal{O}(n\sigma \log \sigma + \mathsf{output})$ time, where output *is the number of p-squares reported.*

By the above discussion, all p-square substrings that are distinct as strings can be computed in $\mathcal{O}(n(\sigma + 1)!)$ time. The key ingredient in the algorithm behind Theorem 2 is a relation between p-squares and σ-mismatch squares (and uniform σ-runs).

1.3 Generalised Squares under Substring Consistent Equivalence Relations

We then show an alternative algorithm for counting p-squares whose complexity does not depend on the alphabet size σ. The algorithm is stated for squares under any *substring consistent equivalence relation* (*SCER* in short). An SCER is a relation $\approx$ on strings such that $X \approx Y$ implies that (1) $|X| = |Y|$ and (2) $X[i\mathinner{.\,.}j] \approx Y[i\mathinner{.\,.}j]$ for all $1 \le i \le j \le |X|$; see [42]. Known examples of SCERs include parameterized matching, order-preserving matching [29, 37], Cartesian tree matching [44], and palindrome pattern matching [26]. Two strings X and Y *order-preserving match* if $|X| = |Y|$, sets $\mathsf{Alph}(X)$ and $\mathsf{Alph}(Y)$ are totally ordered, and there is a bijection $f : \mathsf{Alph}(X) \mapsto \mathsf{Alph}(Y)$ that is increasing (i.e., if $x < y$ then $f(x) < f(y)$) such that $f(X) = Y$. Strings X, Y *Cartesian-tree match* if they have the same shape of a Cartesian tree (cf. [54]). Finally, strings X and Y *palindrome match* if for all $1 \le i \le j \le |X|$, $X[i\mathinner{.\,.}j]$ is a palindrome if and only if $Y[i\mathinner{.\,.}j]$ is a palindrome.

An $\approx$*-square* is a string XY such that $X \approx Y$ under SCER $\approx$. Thus p-squares and order-preserving squares [30] (op-squares, in short) as well as Cartesian-tree squares [44] (CT-squares) and squares in the sense of palindrome matching (palindrome-squares) are $\approx$-squares for the respective SCERs $\approx$. See Figure 2 for an example.

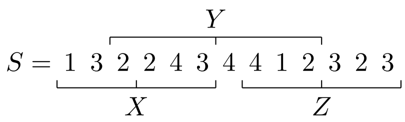

Figure 2 For string $S = 1322434412323$, $X = 132\ 243$ is an op-square (hence, automatically, a p-square and a CT-square), $Y = 2243\ 4412$ is a p-square, and $Z = 412\ 323$ is a CT-square. Among the three substrings, only Z is *not* a palindrome-square, as its second half is a palindrome whereas its first half is not.

In a prefix consistent equivalence relation (PCER), condition (2) of an SCER only needs to hold for $i = 1$. An SCER is a PCER. We say that $\mathcal{E} : \Sigma^* \to \mathbb{Z}$ is an *encoding function* if

$$X \approx Y \quad \Leftrightarrow \quad (\,|X| = |Y| \;\wedge\; \forall_{i\in[1\,..\,|X|]}\, \mathcal{E}(X[1\,..\,i]) = \mathcal{E}(Y[1\,..\,i])\,). \tag{1}$$

▶ **Observation 3.** *For every PCER $\approx$ on strings over integer alphabet there exists an encoding function.*

Proof. Let $\mathcal{E}(X)$ be defined as the number of the equivalence class under $\approx$ of X among all strings of length $|X|$. Let us verify that $\mathcal{E}$ satisfies equivalence (1). ($\Rightarrow$) If $X \approx Y$, then, by definition, $|X| = |Y|$ and $X[1\,..\,i] \approx Y[1\,..\,i]$ for all $i \in [1\,..\,|X|]$. Then indeed $\mathcal{E}(X[1\,..\,i]) = \mathcal{E}(Y[1\,..\,i])$. ($\Leftarrow$) Taking $i = |X| = |Y|$, we obtain $\mathcal{E}(X) = \mathcal{E}(Y)$. As $|X| = |Y|$, we have $X \approx Y$. ◀

The encoding function defined in the proof of Observation 3 could be hard to compute efficiently for a particular PCER $\approx$. It is also stronger, as it satisfies $X \approx Y \Leftrightarrow (\,|X| = |Y| \;\wedge\; \mathcal{E}(X) = \mathcal{E}(Y))$. Luckily, for the aforementioned known SCERs efficient encoding functions are known, as shown in the following Example 4.

For an encoding function $\mathcal{E}$, let $\pi^{\mathcal{E}}(n)$, $\rho^{\mathcal{E}}(n)$ be integer sequences such that for a string T of length n over integer alphabet, after $\pi^{\mathcal{E}}(n)$ preprocessing time, $\mathcal{E}(T[i\,..\,j])$ for any i, j can be computed in $\rho^{\mathcal{E}}(n)$ time.

▶ **Example 4.** For parameterized matching, there exists a classic *prev*-encoding, see [2], that is an encoding function:

$$\mathit{prev}(X) = \begin{cases} |X| - i & X[i] = X[|X|] \text{ and } X[j] \neq X[|X|] \text{ for all } j \in [i+1\,..\,|X|) \\ 0 & \text{otherwise} \end{cases}$$

The *prev*-encodings of all prefixes of a string T can be computed by sorting all pairs $(T[i], i)$. If T is over an integer alphabet, this is performed in $\pi^{prev}(n) = \mathcal{O}(n)$ time. Then $\mathit{prev}(T[i\,..\,j])$ can be computed from $\mathit{prev}(T[1\,..\,j])$ in $\rho^{prev}(n) = \mathcal{O}(1)$ time.

For order-preserving matching, one can use an encoding as pairs $(\alpha(X), \beta(X))$. Here

$$\alpha(X) \text{ is the largest } j < |X| \text{ such that } X[j] = \max\{X[k] \;:\; k \in [1\,..\,|X|),\ X[k] \le X[|X|]\},$$

and if there is no such j, then $\alpha(X) = 0$. Similarly,

$$\beta(X) \text{ is the largest } j < |X| \text{ such that } X[j] = \min\{X[k] \;:\; k \in [1\,..\,|X|),\ X[k] \ge X[|X|]\},$$

and $\beta(X) = 0$ if no such j exists; see [13, 29, 37]. As we require the encoding function to return integers, we can set $\mathcal{E}(X) = \alpha(X) \cdot |X| + \beta(X)$, since $\alpha(X), \beta(X) \in [0\mathinner{.\,.}|X|)$. Then [13, Lemma 4] implies that $\mathcal{E}$ is an encoding function for order-preserving matching and [13, Lemma 24] gives $\pi^{\mathcal{E}}(n) = \mathcal{O}(n\sqrt{\log n})$, $\rho^{\mathcal{E}}(n) = \mathcal{O}(\log n / \log\log n)$.

For Cartesian-tree matching, [44, Theorem 1] shows that the following *parent-distance* representation:

$$PD(X) = \begin{cases} |X| - \max_{1 \le j < |X|}\{j \,:\, X[j] \le X[|X|]\} & \text{if such } j \text{ exists} \\ 0 & \text{otherwise} \end{cases}$$

is an encoding function. Encodings of all prefixes of a string T can be computed in $\pi^{PD}(n) = \mathcal{O}(n)$ time using a folklore nearest smaller value algorithm. In [44, Section 5.2] it is noted that $PD(T[i\mathinner{.\,.}j])$ can be computed from $PD(T[1\mathinner{.\,.}j])$ in $\rho^{PD}(n) = \mathcal{O}(1)$ time.

By X^R we denote the reverse of string X. In [26, Lemma 2] it is shown that the length of the longest suffix palindrome of X, formally:

$$Lpal(X) = \max\{|X| - k + 1 \,:\, X[k\mathinner{.\,.}|X|] = X[k\mathinner{.\,.}|X|]^R\},$$

is an encoding function for palindrome matching.

Computing encodings of substrings $Lpal(T[i\mathinner{.\,.}j])$ can be reduced in linear time to 2D range successor queries as follows.

In the 2D range successor problem we are given n points in an $n \times n$ grid and we are to answer queries that, given a rectangle in the grid, report a point in the rectangle with the smallest first coordinate, if any. Such queries can be answered in $\mathcal{O}(\log\log n)$ time after $\mathcal{O}(n\sqrt{\log n})$ preprocessing [18].

In the reduction, we compute all maximal palindromes in T in $\mathcal{O}(n)$ time using Manacher's algorithm [41]. Let us show how we deal with odd-length palindromes; the approach for even-length palindromes is analogous. For a maximal palindrome $T[c-r\mathinner{.\,.}c+r]$ with $c \in [1\mathinner{.\,.}n]$ and $r \ge 0$, we create a point $(c, c+r)$. To compute $Lpal(T[i\mathinner{.\,.}j])$, it suffices to find the point in the rectangle $[\lceil (i+j)/2 \rceil \mathinner{.\,.} j] \times [j\mathinner{.\,.}\infty)$ with the smallest first coordinate. If x is the sought coordinate, the longest odd-length suffix palindrome of $T[i\mathinner{.\,.}j]$ has length $2(j-x)+1$.

By [18], we get $\pi^{Lpal}(n) = \mathcal{O}(n\sqrt{\log n})$ and $\rho^{Lpal}(n) = \mathcal{O}(\log\log n)$.

In Section 5 we show the next theorem on efficient counting of $\approx$-squares.

▶ **Theorem 5.** *Let $\approx$ be an SCER and $\mathcal{E}$ be its encoding function. The number of non-equivalent $\approx$-square substrings in a length-n string can be computed in $\mathcal{O}(\pi^{\mathcal{E}}(n) + n\rho^{\mathcal{E}}(n) + n\log n)$ time.*

The number of $\approx$-square substrings in a length-n string that are distinct as strings can be computed in the same time complexity.

Consequently, in a length-n string, the numbers of non-equivalent p-squares, op-squares, CT-squares, and palindrome-squares, as well as the numbers of the respective squares that are distinct as strings can be computed in $\mathcal{O}(n\log n)$ time, as the respective SCERs enjoy encoding functions with $\pi^{\mathcal{E}}(n) = \mathcal{O}(n\log n)$ and $\rho^{\mathcal{E}}(n) = \mathcal{O}(\log n)$ as shown in Example 4. In particular, the algorithm of Theorem 5 in the case that $\sigma = \omega(\log n)$ improves upon the $\mathcal{O}(n\sigma)$-time algorithm of Gawrychowski, Ghazawi, and Landau [19] for reporting (thus, counting) op-squares that are distinct as strings. Moreover, the obtained complexity for counting non-equivalent p-squares (p-squares that are distinct as strings, respectively) is better than the one in Theorem 2 if $\sigma = \omega(\log n / \log\log n)$ ($\sigma = \Omega(\log\log n)$, respectively).

Theorem 5 involves constructing a $\approx$-suffix tree data structure (see Section 3 for a definition).

2 Upper Bound on the Number of k-Runs

This section introduces an upper bound on the number of k-runs (proof of Theorem 1) which implies their efficient computation by the algorithm of Kolpakov and Kucherov [34, 35]. A position i in string T is called an *ℓ-mismatching position* if $i \in [1\,..\,n-\ell]$ and $T[i] \neq T[i+\ell]$. In particular, $T[i\,..\,i+2\ell)$ is a k-mismatch square if and only if $T[i\,..\,i+\ell)$ contains at most k ℓ-mismatching positions. Let us formally define a uniform k-run.

▶ **Definition 6.** *A substring $T[a\,..\,b)$ is called a* uniform k-run of period ℓ *if $b-a \geq 2\ell$ and the sets $\{j \in [i\,..\,i+\ell) : T[j] \neq T[j+\ell]\}$ of ℓ-mismatching positions for all $i \in [a\,..\,b-2\ell]$ have cardinality at most k and are all the same. We are interested in maximal uniform k-runs.*

A *gapped repeat* in a string T is a fragment of the form UVU, for $|U| > 0$. Fragments denoted by U are called *arms* of the gapped repeat (left and right arm). The *period* of the gapped repeat is defined as $|UV|$. An α-gapped repeat (for $\alpha \geq 1$) in T is a gapped repeat UVU such that $|UV| \leq \alpha|U|$. An (α-)gapped repeat is called *maximal* if its arms cannot be extended simultaneously with the same character to the right or to the left. A maximal (α-)gapped repeat will be called an (α-)MGR, for short. Gawrychowski, I, Inenaga, Köppl, and Manea [20] showed that, for a real $\alpha \geq 1$, a string of length n contains $\mathcal{O}(n\alpha)$ α-MGRs.

A *generalised run* is a pair $(T[x\,..\,y), p)$ such that p satisfies $2p \leq y-x$ and (1) $x = 1$ or $T[x-1\,..\,y)$ does not have period p, and (2) $y = |T|+1$ or $T[x\,..\,y]$ does not have period p. That is, a generalised run is a 0-run. A run is a generalised run in which p is the smallest period of $T[x\,..\,y)$. A string of length n contains less than n runs and $\mathcal{O}(n)$ generalised runs [3].

▶ **Definition 7.** *We say that an MGR $UVU = T[x\,..\,y)$ with period $\ell = |UV|$ in T* induces *a uniform k-run $T[a\,..\,b)$ of period ℓ if $[x\,..\,y-\ell) \cap [a\,..\,b-\ell) \neq \emptyset$.*

We further say that a generalised run $(T[x\,..\,y), \ell)$ induces *a uniform k-run $T[a\,..\,b)$ of period ℓ if $[x\,..\,y-\ell) \cap [a\,..\,b-\ell) \neq \emptyset$.*

Let us emphasize that the uniform k-run does not need to be a substring of the MGR or generalised run. Intuitively, an MGR induces a uniform k-run if the left half of at least one k-mismatch square of the k-run contains a position of the left arm of the MGR; see Figure 3. If an MGR or a generalised run induces a uniform k-run, we say that the k-run is *induced* by the MGR or run, respectively. We also say that the MGR or generalised run induces all k-mismatch squares in the uniform k-run that it induces.

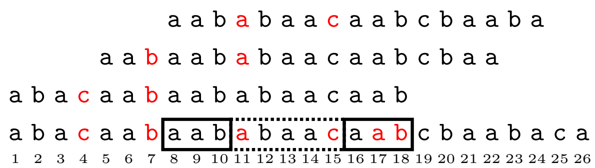

Figure 3 String $T[1\,..\,26]$ from Figure 1 with 8-mismatching positions shown in red. The string contains three uniform 2-runs with period 8, $T[1\,..\,18]$, $T[5\,..\,22]$, and $T[8\,..\,24]$ (in Figure 1, the last two form a single 2-run). They are all induced by a 3-MGR UVU for $U = T[8\,..\,10] = T[16\,..\,18]$ and $V = T[11\,..\,15]$. In the proof of Lemma 8 for this MGR, we have $L = \{7, 4, 0\}$ and $R = \{11, 15, 17\}$.

▶ **Lemma 8.** *An MGR induces at most $2k+1$ uniform k-runs.*

Proof. Consider an MGR $T[x\,..\,y)$ with period ℓ. Let R be the set of $k+1$ smallest ℓ-mismatching positions in $[y-\ell\,..\,y)$ in T; if there are no such $k+1$ positions, we select all ℓ-mismatching positions in $[y-\ell\,..\,y)$ in T to the set R. Similarly, let L be the set of $k+1$ largest ℓ-mismatching positions in $[x-\ell+1\,..\,x)$ in T; if there are no $k+1$ such positions, we select all ℓ-mismatching positions in $[x-\ell+1\,..\,x)$. If $|R| \le k$, we insert y to R. If $|L| \le k$, we insert $x-\ell$ to L. This way, if a k-mismatch square $T[i\,..\,i+2\ell)$ is induced by the MGR $T[x\,..\,y)$, then $i \in I$ for $I = [\min L + 1\,..\,\max R - \ell]$ (as otherwise the left half of the square would contain more than k ℓ-mismatching positions or would not touch $T[x\,..\,y-\ell)$).

Let us consider a window of length ℓ sliding left-to-right over T with the left end point of the window located in interval I. There are at most $2k$ moments when the number of ℓ-mismatching positions in the window changes. Between them, we obtain uniform k-runs if the number of ℓ-mismatching positions in the window is at most k. ◀

An analogous lemma for generalised runs is obtained with the same proof (replacing "MGR" by "generalised run"); see Figure 4.

Figure 4 String $T[1\,..\,26]$ with its 6-mismatching positions shown in red. A generalised run $T[6\,..\,21]$ with period 6 (rectangle) induces five uniform 2-runs with period 6 shown above. In this example, the generalised run corresponds to a run with period 3.

▶ **Lemma 9.** *A generalised run induces at most $2k+1$ uniform k-runs.*

A uniform k-run can be induced by several MGRs and generalised runs.

▶ **Example 10.** The middle uniform 2-run $T[5\,..\,22]$ in Figure 3 is induced also by two other MGRs with period 8, $T[5\,..\,14]$ (with arms $T[5\,..\,6] = T[13\,..\,14] =$ aa) and $T[12\,..\,22]$ (with arms $T[12\,..\,14] = T[20\,..\,22] =$ baa).

For a gapped repeat UVU, the fraction $\frac{|UV|}{|U|}$ is called the *gap ratio*. That is, the gap ratio is the minimum α such that UVU is an α-gapped repeat. We assign to each gapped repeat UVU in T a *weight* equal to the reciprocal of its gap ratio, that is, $\frac{|U|}{|UV|}$.

▶ **Lemma 11.** *If $\ell \ge 4k$ and $T[a\,..\,b)$ is a uniform k-run of period ℓ, then it is induced by a generalised run or by some number of $(2k+2)$-MGRs of total weight at least $\frac{1}{4}$.*

Proof. Let $p_1, \ldots, p_t$ be all ℓ-mismatching positions among $[a\,..\,b-\ell)$, listed in an increasing order. We define sentinel p_0 being the maximum ℓ-mismatching position that is smaller than a; if there is no such position, we set $p_0 = 0$. Similarly, we define sentinel p_{t+1} as the minimum ℓ-mismatching position that is at least $b-\ell$; if no such position exists, we set $p_{t+1} = n+1-\ell$.

Let us fix $s \in [0\,..\,t]$. We have $U_s := T[p_s+1\,..\,p_{s+1}) = T[p_s+\ell+1\,..\,p_{s+1}+\ell)$, so $W_s := T[p_s+1\,..\,p_{s+1}+\ell)$ has period ℓ. Moreover, we have (1) $p_s = 0$ or $T[p_s] \ne T[p_s+\ell]$ and (2) $p_{s+1}+\ell = n+1$ or $T[p_{s+1}] \ne T[p_{s+1}+\ell]$. If $0 < |U_s| < \ell$, then W_s is an MGR with arms equal to U_s. If $|U_s| \ge \ell$, W_s is a generalised run. The MGR or generalised run induces $T[a\,..\,b)$ unless $p_{s+1} = a$ or $p_s = b-\ell-1$.

Assume that the uniform k-run $T[a\,..\,b)$ is not induced by a generalised run. If a string W_s, for $s \in [0\,..\,t]$, is not a $(2k+2)$-MGR of period ℓ, the length of its left arm U_s is smaller than $\frac{\ell}{2k+2}$. Hence, for strings W_s that are not $(2k+2)$-MGRs, the total length of their left arms U_s is at most $\frac{\ell}{2}$. The remaining $(2k+2)$-MGRs among W_s have total length of left arms at least $(b-a-\ell) - \frac{\ell}{2} - k \geq \frac{\ell}{2} - k$ (accounting for the at most k ℓ-mismatching positions $p_1, \ldots, p_t$) which is at least $\frac{\ell}{4}$ by the assumption of the lemma. This means that the total weight of $(2k+2)$-MGRs that induce the k-run, defined as the sum of the lengths of their left arms divided by their periods, all equal to ℓ, is at least $\frac{1}{4}$. ◀

We show a theorem that implies Theorem 1.

▶ **Theorem 12.** *A string of length n contains $\mathcal{O}(nk \log k)$ uniform k-runs.*

Proof. Let T be a string of length n. Let us consider a bipartite graph $G = (V_1 \cup V_2, E)$ such that vertices in V_1 are uniform k-runs in T, vertices in V_2 are $(2k+2)$-MGRs and generalised runs in T, and there is an edge $uw \in E$, for $u \in V_1$ and $w \in V_2$, if k-run u is induced by the MGR or generalised run w. Our goal is to bound $|V_1|$ from above.

Let us remove from G all vertices in V_1 that are uniform k-runs with period at most $4k$. There are at most $4kn$ such k-runs (as there are at most $4kn$ substrings of T of length at most $4k$). Let us also remove from G all vertices in V_2 that are generalised runs and all vertices in V_1 that are adjacent to at least one of the removed vertices in V_2. By [3], this way at most $1.5n$ vertices are removed from V_2, so by Lemma 9, $\mathcal{O}(nk)$ vertices are removed from V_1. Let $G' = (V_1' \cup V_2', E')$ be the remaining graph. We assume that each edge has a weight equal to the weight of the MGR in V_2' it is incident to. In order to bound $|V_1'|$, we will consider the *total weight of edges* in E'.

For each $\alpha \in [2\,..\,2k+2]$, let x_α be the number of α-MGRs in T that are not $(\alpha-1)$-MGRs. Each such MGR has weight at most $\frac{1}{\alpha-1}$ and $2k+1 \leq 3k$ edges incident to it (cf. Lemma 8). Thus the sum of weights of edges in E' is bounded from above by

$$\sum_{\alpha=2}^{2k+2} \frac{3kx_\alpha}{\alpha-1} \tag{2}$$

The improved upper bound on the number of α-MGRs of I and Köppl [27] implies that

$$\sum_{i=2}^{\alpha} x_i < 13n\alpha \quad \text{for every } \alpha \in [2\,..\,2k+2] \tag{3}$$

We show how to bound the number (2) in general.

▷ Claim 13. For every sequence $(x_2, \ldots, x_{2k+2})$ of non-negative integers that satisfies the condition (3), the value (2) is $\mathcal{O}(nk \log k)$.

Proof. As long as there exists $\alpha \in [3\,..\,2k+2]$ such that $x_\alpha > 13n$, we select $i \in [2\,..\,\alpha)$ such that $x_i < 13n$ or $i = 2$ and $x_i < 26n$, decrement x_α and increment x_i. Such i exists by (3) and the pigeonhole principle. The operation does not change any lhs in (3) and increases (2).

In the end, we have $x_2 \leq 26n$ and $x_\alpha \leq 13n$ for all $\alpha \in [3\,..\,2k+2]$. Hence, (2) is bounded as:

$$\sum_{\alpha=2}^{2k+2} \frac{3kx_\alpha}{\alpha-1} \;\leq\; 78k + 39nk \sum_{\alpha=2}^{2k+1} \frac{1}{\alpha} \;\leq\; 78k + 39nk(\ln(2k+1)+1) \;=\; \mathcal{O}(nk \log k). \qquad ◁$$

By the claim, the sum of weights of edges in G' is $\mathcal{O}(nk \log k)$. By Lemma 11, for each uniform k-run with period $\ell \geq 4k$ that is not induced by a generalised run, the total weight of edges incident to it is at least $\frac{1}{4}$. This concludes that $|V_1'| = \mathcal{O}(nk \log k)$, so the total number of uniform k-runs (across all periods $\ell \in [1\,..\,\lfloor n/2 \rfloor]$) is $|V_1| = \mathcal{O}(nk \log k)$. ◀

A k-run of period ℓ contains a prefix being a uniform k-run of period ℓ, constructed by taking k-mismatch squares of length 2ℓ at subsequent positions as long as their sets of ℓ-mismatching positions are the same. By maximality, no two k-runs with equal period start at the same position, so this way each k-run produces a different uniform k-run. Thus the number of k-runs in T does not exceed the number of uniform k-runs, which proves Theorem 1. By the same argument, each k-repetition implies a unique uniform k-run and so the number of k-repetitions is $\mathcal{O}(nk \log k)$. (We refer the reader to the definition of k-repetition in [34, 35, 38] as this notion is not used below.)

3 ≈-Suffix Trees and Their Applications

We introduce useful tools for the next two sections. Cole and Hariharan [11] defined a *quasi-suffix collection* as a collection of strings $S_1, S_2, \ldots, S_n$ that satisfies the following conditions:

1. $|S_1| = n$ and $|S_i| = |S_{i-1}| - 1$.
2. No S_i is a prefix of another S_j.
3. Suppose strings S_i and S_j have a common prefix of length $\ell > 0$. Then S_{i+1} and S_{j+1} have a common prefix of length at least $\ell - 1$.

A quasi-suffix collection is specified implicitly by a character oracle that given i, j returns $S_i[j]$. They obtain the following result.

▶ **Theorem 14** ([11]). *The compacted trie of a quasi-suffix collection of n strings can be constructed in $\mathcal{O}(n)$ expected time assuming that the character oracle works in $\mathcal{O}(1)$ time.*

For a string T of length n, the *≈-suffix tree* of T is a compacted trie of strings $\mathit{Code}(T[i\,..\,n])\#$, $i \in [1\,..\,n+1]$, where $\mathit{Code}(X) = \mathcal{E}(X[1])\mathcal{E}(X[1\,..\,2]) \cdots \mathcal{E}(X)$ for a string X and $\#$ is an end-marker that is not an encoding of any string. We extend the sequences π, ρ so that for any string $T \in [0\,..\,\sigma)^n$, after $\pi^{\mathcal{E}}(n, \sigma)$ preprocessing time, $\mathcal{E}(T[i\,..\,j])$ for any i, j can be computed in $\rho^{\mathcal{E}}(n, \sigma)$ time. By $\gamma^{\mathcal{E}}(n, \sigma)$ we denote an upper bound on the total number of distinct characters in the strings stored in the ≈-suffix tree of a string in $[0\,..\,\sigma)^n$. Theorem 14 implies the following corollary. (A similar result, but only for order-preserving equivalence, was shown in [19, Lemma 16].)

▶ **Corollary 15.** *Let ≈ be an SCER and $\mathcal{E}$ be its encoding function. The ≈-suffix tree of a string in $[0\,..\,\sigma)^n$ can be constructed in worst case time:*

$$\mathcal{O}(\pi^{\mathcal{E}}(n, \sigma) + n\rho^{\mathcal{E}}(n, \sigma) + \min\{n \log^2 \log n / \log \log \log n,\ n\gamma^{\mathcal{E}}(n, \sigma)\}).$$

Proof. Let us verify that strings $\mathit{Code}(T[i\,..\,n])\#$ satisfy the conditions 1-3 of a quasi-suffix collection. Condition 1 for $n+1$ is obvious. Condition 2 follows due to the end-marker. As for condition 3, assume $\mathsf{LCP}(S_i, S_j) = \ell > 0$ and let $i \neq j$ as otherwise the conclusion is trivial. Then $\ell < |S_i|, |S_j|$ because of the end-marker. By equivalence (1), we have $T[i\,..\,i+\ell) \approx T[j\,..\,j+\ell)$. Because ≈ is an SCER, we have $T[i+1\,..\,i+\ell) \approx T[j+1\,..\,j+\ell)$. Hence, $\mathsf{LCP}(S_{i+1}, S_{j+1}) \geq \ell - 1$ by equivalence (1).

An oracle for the quasi-suffix collection S_i answers queries in $\mathcal{O}(\rho^{\mathcal{E}}(n, \sigma))$ time after $\mathcal{O}(\pi^{\mathcal{E}}(n, \sigma))$ preprocessing. The only source of randomness in the algorithm behind Theorem 14 is the need to maintain, for each explicit node of the current tree, a dictionary indexed by the next character on an outgoing edge. If we store the respective characters per each node in a dynamic predecessor data structure of Andersson and Thorup [1], the total space remains $\mathcal{O}(n)$ and each predecessor query is answered in $\mathcal{O}(\log^2 \log n/\log\log\log n)$ time in the worst case. Alternatively, one can store all the children of a node in a list, to achieve $\mathcal{O}(\gamma^{\mathcal{E}}(n,\sigma))$ space per an explicit node and $\mathcal{O}(\gamma^{\mathcal{E}}(n,\sigma))$ time for a predecessor query. The complexity follows. ◀

▶ Remark 16. For SCERs mentioned in Example 4, we obtain the following time complexities for constructing $\approx$-suffix trees in the respective settings: $\mathcal{O}(n\log^2\log n/\log\log\log n)$ for parameterized matching and Cartesian tree matching, $\mathcal{O}(n\sqrt{\log n})$ for palindrome matching (that matches the complexity from [26]), and $\mathcal{O}(n\log n/\log\log n)$ for order-preserving matching (a faster, $\mathcal{O}(n\sqrt{\log n})$-time construction was proposed in [13]).

For two strings X and Y, by $\mathsf{LCP}^{\approx}(X,Y)$ we denote $\max\{\ell \ge 0 : X[1\mathinner{.\,.}\ell] \approx Y[1\mathinner{.\,.}\ell]\}$. As in the case of standard suffix trees, by equivalence (1), having an $\approx$-suffix tree of T and the data structure answering lowest common ancestor queries for nodes in $\mathcal{O}(1)$ time after $\mathcal{O}(n)$ preprocessing [25, 5], we can answer $\mathsf{LCP}^{\approx}$ queries about pairs of substrings of T in $\mathcal{O}(1)$ time.

The longest previous $\approx$-factor array $\mathsf{LPF}^{\approx}$ for a string T is an array such that

$$\mathsf{LPF}^{\approx}[i] = \max\{\ell \ge 0 : T[i\mathinner{.\,.}i+\ell) \approx T[j\mathinner{.\,.}j+\ell) \text{ for some } j \in [1\mathinner{.\,.}i)\}.$$

Computing this array can be stated in terms of the $\approx$-suffix tree: for a leaf with index i of the tree we are to find a leaf with index $j < i$ such that the lowest common ancestor of the two leaves is as low as possible. The actual value $\mathsf{LPF}^{\approx}[i]$ is then the (weighted) depth of this ancestor. In [13, Theorem 16] it was shown that this problem, stated for an arbitrary rooted (weighted) tree with n leaves, can be solved in $\mathcal{O}(n)$ time. Thus, the $\mathsf{LPF}^{\approx}$ array for a length-n string can be computed in $\mathcal{O}(n)$ time if the $\approx$-suffix tree is available.

4 Counting p-Squares in $\mathcal{O}(n\sigma\log\sigma)$ Time

In this section we consider $T \in [0\mathinner{.\,.}\sigma)^n$ and $\approx$ denotes the relation of parameterized matching. We use the following function $\mathbf{E}$ for $\approx$ (see Example 17):

$$\mathbf{E}(X) = |\mathsf{Alph}(U)| \text{ where } U \text{ is the longest suffix of } X[1\mathinner{.\,.}|X|) \text{ without letter } X[|X|].$$

▶ **Example 17.** $\mathbf{E}(X)$ for $X = [1,2,1,1,2,3,2,1,4]$ is as follows:

i	1	2	3	4	5	6	7	8	9
$X[i]$	1	2	1	1	2	3	2	1	4
$\mathbf{E}(X[1\mathinner{.\,.}i])$	0	1	1	0	1	2	1	2	3

A similar function was used in the context of parameterized matching in [30]. Lemma 18 below shows that $\mathbf{E}$ is an encoding function for parameterized matching and Lemma 19 shows that it can be computed efficiently.

The ($\Leftarrow$) part of the proof of Lemma 18 is similar to the proof of [30, Lemma 5.4]. A proof of the lemma can be found in the full version.

▶ **Lemma 18.** $\mathbf{E}$ *is an encoding function for parameterized matching.*

▶ **Lemma 19.** *We have* $\pi^{\mathbf{E}}(n,\sigma) = \mathcal{O}(n\sigma)$, $\rho^{\mathbf{E}}(n,\sigma) = \mathcal{O}(\sigma)$, *and* $\gamma^{\mathbf{E}}(n,\sigma) = \sigma + 1$.

Proof. Let $T \in [0\,..\,\sigma)^n$. By definition, $\mathbf{E}(T[i\,..\,j]) \in [0\,..\,\sigma)$ for any indices i, j. Hence, $\gamma^{\mathbf{E}}(n,\sigma) = \sigma + 1$ (including the sentinel). In order to compute encodings of substrings of T, we can store for every $c \in [0\,..\,\sigma)$, the numbers of characters c in respective prefixes of T. Moreover, for every index i, we store the position $\mathit{prev}[i] \in [0\,..\,i)$ of the last occurrence of character $T[i]$ in $T[1\,..\,i)$; $\mathit{prev}[i] = 0$ if there is no such occurrence. The array prev can be computed from left to right in $\mathcal{O}(n+\sigma)$ time by storing an array of size σ of rightmost occurrences of each character. Then indeed $\pi^{\mathbf{E}}(n,\sigma) = \mathcal{O}(n\sigma)$. When computing $\mathbf{E}(T[i\,..\,j])$, we can compare the counts of every character c at prefixes $T[1\,..\,j)$ and $T[1\,..\,i')$ where $i' = \max(\mathit{prev}[j]+1, i)$. Hence, $\rho^{\mathbf{E}}(n,\sigma) = \mathcal{O}(\sigma)$. ◀

We define strings $\overrightarrow{T}$, $\overleftarrow{T}$ of length n such that

$$\overrightarrow{T}[i] = \mathbf{E}(T[1\,..\,i]), \quad \overleftarrow{T}[i] = \mathbf{E}((T[i\,..\,n])^R).$$

Our goal is to reduce reporting non-equivalent p-squares to computing uniform k-runs. We use two auxiliary lemmas.

▶ **Lemma 20.** *If $T[i\,..\,i{+}2\ell)$ is a p-square, then $\overrightarrow{T}[i\,..\,i{+}2\ell)$ and $\overleftarrow{T}[i\,..\,i{+}2\ell)$ are σ-mismatch squares.*

Proof. We show a proof that $\overrightarrow{T}[i\,..\,i{+}2\ell)$ is a σ-mismatch square; the proof that $\overleftarrow{T}[i\,..\,i{+}2\ell)$ is a σ-mismatch square is symmetric.

Let $i_1, \ldots, i_t \in [0\,..\,\ell)$, for $t \in [1\,..\,\sigma]$, be the positions of leftmost occurrences of characters from $[0\,..\,\sigma)$ in $T[i\,..\,i+\ell)$; if some character is not present in $T[i\,..\,i+\ell)$, it is not included in the sequence. Let $j \in [0\,..\,\ell) \setminus \{i_1, \ldots, i_t\}$. Let $j' \in [0\,..\,j)$ be the maximum index such that $T[i+j'] = T[i+j]$, so $T[i+\ell+j'] = T[i+\ell+j]$ and $T[i+\ell+j''] \neq T[i+\ell+j]$ for all $j'' \in [j'+1\,..\,j)$ because $T[i\,..\,i+\ell) \approx T[i+\ell\,..\,i+2\ell)$. Then

$$\overrightarrow{T}[i+j] = |\mathsf{Alph}(T[i+j'+1\,..\,i+j))| = |\mathsf{Alph}(T[i+\ell+j'+1\,..\,i+\ell+j))| = \overrightarrow{T}[i+\ell+j]$$

by the fact that $\approx$ is an SCER. Hence, $\overrightarrow{T}[i\,..\,i+\ell)$ and $\overrightarrow{T}[i+\ell\,..\,i+2\ell)$ have at most $t \leq \sigma$ mismatches, as required. ◀

▶ **Lemma 21.** *If $T[i\,..\,i+2\ell)$ is a p-square and $\overrightarrow{T}[i+\ell] = \overrightarrow{T}[i+2\ell]$, then $T[i+1\,..\,i+2\ell]$ is a p-square. Similarly, if $T[i\,..\,i+2\ell)$ is a p-square and $\overleftarrow{T}[i-1] = \overleftarrow{T}[i+\ell-1]$, then $T[i-1\,..\,i+2\ell-1)$ is a p-square.*

Proof. Again, we only prove the first part of the lemma. We have $T[i\,..\,i{+}\ell) \approx T[i{+}\ell\,..\,i{+}2\ell)$, so $T[i+1\,..\,i+\ell) \approx T[i+1+\ell\,..\,i+2\ell)$ since $\approx$ is an SCER. If

$$\overrightarrow{T}[i+2\ell] = \overrightarrow{T}[i+\ell] < |\mathsf{Alph}(T[i+1\,..\,i+\ell))| = |\mathsf{Alph}(T[i+1+\ell\,..\,i+2\ell))|,$$

then $\overrightarrow{T}[i+\ell] = \mathbf{E}(T[i+1\,..\,i+\ell])$, $\overrightarrow{T}[i+2\ell] = \mathbf{E}(T[i+1+\ell\,..\,i+2\ell])$, and $T[i+1\,..\,i+\ell] \approx T[i+1+\ell\,..\,i+2\ell]$ by the fact that $\mathbf{E}$ is an encoding function for parameterized matching (Lemma 18). Otherwise,

$$T[i+\ell] \notin \mathsf{Alph}(T[i+1\,..\,i+\ell)) \quad \text{and} \quad T[i+2\ell] \notin \mathsf{Alph}(T[i+1+\ell\,..\,i+2\ell)),$$

so we immediately obtain $T[i+1\,..\,i+\ell] \approx T[i+1+\ell\,..\,i+2\ell]$. In both cases, $T[i+1\,..\,i+2\ell]$ is a p-square. ◀

If $T[a\,..\,b)$ is a uniform k-run with period ℓ, then there is no ℓ-mismatching position in $T[a+\ell\,..\,b-\ell)$, i.e., $T[a+\ell\,..\,b-\ell) = T[a+2\ell\,..\,b)$. Hence, if $\overrightarrow{T}[a\,..\,b)$ is a uniform k-run with period ℓ and $T[a\,..\,a+2\ell)$ is a p-square, then by Lemma 21, each string $T[i\,..\,i+2\ell)$ for $i \in [a\,..\,b-2\ell]$ is a p-square. With this intuition, we are ready to obtain the reduction.

▶ **Lemma 22.** *Let $T \in [0\,..\,\sigma)^n$. Reporting non-equivalent p-squares in T reduces in $\mathcal{O}(n\sigma+r)$ time to computing uniform σ-runs in $\overrightarrow{T}$ and $\overleftarrow{T}$, where r is the number of these σ-runs.*

Reporting p-squares in T that are distinct as strings reduces to the same problem in $\mathcal{O}(n\sigma + r + \mathsf{output})$ time, where output is the number of p-squares reported.

Proof. For a uniform k-run $T[a\,..\,b)$ of period ℓ, we define its *interval* as $[a\,..\,b-2\ell]$. For each $\ell \in [1\,..\,\lfloor n/2 \rfloor]$, let $\overrightarrow{\mathcal{P}}_\ell$ and $\overleftarrow{\mathcal{P}}_\ell$ be sets of intervals of uniform σ-runs of period ℓ in $\overrightarrow{T}$ and $(\overleftarrow{T})^R$, respectively. Let $\overleftarrow{\mathcal{P}}'_\ell = \{[n-b-2\ell\,..\,n-a-2\ell] : [a\,..\,b] \in \overleftarrow{\mathcal{P}}_\ell\}$. Finally, let $\mathcal{P}_\ell = \left(\bigcup \overrightarrow{\mathcal{P}}_\ell\right) \cap \left(\bigcup \overleftarrow{\mathcal{P}}'_\ell\right)$.

▷ Claim 23. If $T[i\,..\,i+2\ell)$ is a p-square in T, then $i \in \mathcal{P}_\ell$.

Proof. By Lemma 20, if $T[i\,..\,i+2\ell)$ is a p-square in T, then $\overrightarrow{T}[i\,..\,i+2\ell)$ and $\overleftarrow{T}[i\,..\,i+2\ell) = (\overleftarrow{T})^R[n-i-2\ell\,..\,n-i)$ are σ-mismatch squares. Therefore, there exist intervals $[a\,..\,b] \in \overrightarrow{\mathcal{P}}_\ell$ and $[a'\,..\,b'] \in \overleftarrow{\mathcal{P}}_\ell$ such that $i \in [a\,..\,b]$ and $n-i-2\ell \in [a'\,..\,b']$. In particular, $i \in \bigcup \overrightarrow{\mathcal{P}}_\ell$. We have $[n-b'-2\ell\,..\,n-a'-2\ell] \in \overleftarrow{\mathcal{P}}'_\ell$ and $i \in [n-b'-2\ell\,..\,n-a'-2\ell]$, so $i \in \bigcup \overleftarrow{\mathcal{P}}'_\ell$. ◁

We store $\mathcal{P}_\ell$ as a union of non-empty intervals of the form $\{I \cap J : I \in \overrightarrow{\mathcal{P}}_\ell, J \in \overleftarrow{\mathcal{P}}'_\ell\}$. Let this representation be denoted as $\mathcal{R}_\ell$.

All the representations $\mathcal{R}_\ell$ can be computed from $\overrightarrow{\mathcal{P}}_\ell$ and $\overleftarrow{\mathcal{P}}'_\ell$ in $\mathcal{O}(n + \sum_\ell(|\overrightarrow{\mathcal{P}}_\ell| + |\overleftarrow{\mathcal{P}}_\ell|))$ total time. Let us bucket sort all endpoints of intervals in $\overrightarrow{\mathcal{P}}_\ell$ and $\overleftarrow{\mathcal{P}}'_\ell$, for each ℓ. When processing the endpoints for a given ℓ, we keep track of the number of intervals containing a given position. This counter never exceeds 2 as intervals in each of $\overrightarrow{\mathcal{P}}_\ell$ and $\overleftarrow{\mathcal{P}}'_\ell$ are pairwise disjoint. Whenever the counter reaches 2, for some endpoint a, it will drop at the next endpoint encountered, say b. Then $[a\,..\,b)$ is inserted into $\mathcal{R}_\ell$.

The next claim shows that, for each interval in $\mathcal{R}_\ell$, all positions correspond to p-squares of length 2ℓ or none of them does.

▷ Claim 24. Let $[a\,..\,b] \in \mathcal{R}_\ell$. For each $i \in [a\,..\,b]$, $T[i\,..\,i+2\ell)$ is a p-square if and only if $T[a\,..\,a+2\ell)$ is a p-square.

Proof. Let us fix $i \in [a\,..\,b]$. Let $[a'\,..\,b'] \in \overrightarrow{\mathcal{P}}_\ell$ and $[a''\,..\,b''] \in \overleftarrow{\mathcal{P}}'_\ell$ be intervals such that $[a\,..\,b] = [a'\,..\,b'] \cap [a''\,..\,b'']$.

Assume first that $T[a\,..\,a+2\ell)$ is a p-square. As $\overrightarrow{\mathcal{P}}_\ell$ was computed from uniform σ-runs, we have $\overrightarrow{T}[a'+2\ell\,..\,b'+2\ell) = \overrightarrow{T}[a'+\ell\,..\,b'+\ell)$, so $\overrightarrow{T}[a+2\ell\,..\,i+2\ell) = \overrightarrow{T}[a+\ell\,..\,i+\ell)$. By Lemma 21 applied for each subsequent position in $[a\,..\,i)$, $T[i\,..\,i+2\ell)$ is a p-square.

Now assume that $T[i\,..\,i+2\ell)$ is a p-square. As $[a''\,..\,b''] \in \overleftarrow{\mathcal{P}}'_\ell$, $[n-b''-2\ell\,..\,n-a''-2\ell] \in \overleftarrow{\mathcal{P}}_\ell$. By definition, we have $(\overleftarrow{T})^R[n-b''\,..\,n-a'') = (\overleftarrow{T})^R[n-b''-\ell\,..\,n-a''-\ell)$, i.e., $\overleftarrow{T}[a''\,..\,b'') = \overleftarrow{T}[a''+\ell\,..\,b''+\ell)$. In particular, $\overleftarrow{T}[a\,..\,i) = \overleftarrow{T}[a+\ell\,..\,i+\ell)$. By Lemma 21 applied for each position in $[a+1\,..\,i]$ in decreasing order, $T[a\,..\,a+2\ell)$ is a p-square. ◁

By Corollary 15 and Lemmas 18 and 19, after $\mathcal{O}(n\sigma)$ preprocessing we can check if a given even-length substring of T is a p-square in $\mathcal{O}(1)$ time using an $\mathsf{LCP}^{\approx}$ query. For each $\ell \in [1\,..\,\lfloor n/2 \rfloor]$ and each interval $[a\,..\,b] \in \mathcal{R}_\ell$, we check if $T[a\,..\,a+2\ell)$ is a p-square. By Claim 24, if so, we obtain an interval of occurrences of p-squares, and if not, then none of the positions in $[a\,..\,b]$ is the start of a p-square of length 2ℓ in T. The total number of intervals in $\mathcal{R}_\ell$ is linear in the number of uniform σ-runs in $\overrightarrow{T}$ and $\overleftarrow{T}$, so we obtain $\mathcal{O}(r)$ intervals of occurrences of p-squares. By Claim 23, we do not miss any p-squares.

The final step of reporting non-equivalent p-squares mimics an analogous algorithm for standard squares of Bannai, Inenaga, and Köppl [4]. We report non-equivalent p-squares at their leftmost occurrence in T and use the $\mathsf{LPF}^{\approx}$ array to identify them in intervals of occurrences. More precisely, we compute the $\mathsf{LPF}^{\approx}$ array in $\mathcal{O}(n\sigma)$ time and construct in $\mathcal{O}(n)$ time a data structure for answering Range Minimum Queries (RmQs) over $\mathsf{LPF}^{\approx}$ in $\mathcal{O}(1)$ time per query; see [5, 25]. For each of the $\mathcal{O}(r)$ intervals $[a\,..\,b]$ of starting positions of p-squares of length 2ℓ, we want to list all positions $i \in [a\,..\,b]$ such that $\mathsf{LPF}^{\approx}[i] < 2\ell$ and report corresponding leftmost occurrences of p-squares $T[i\,..\,i+2\ell)$. We ask an RmQ on $\mathsf{LPF}^{\approx}$ on the interval $[a\,..\,b]$; let the minimum be attained for $i \in [a\,..\,b]$. If $\mathsf{LPF}^{\approx}[i] \geq 2\ell$, the algorithm is finished. Otherwise, we report $T[i\,..\,i+2\ell)$ and run the algorithm recursively on intervals $[a\,..\,i-1]$ and $[i+1\,..\,b]$. The total time complexity is proportional to the number of reported p-squares plus 1. By [24], T contains $\mathcal{O}(n\sigma)$ non-equivalent p-squares. This completes an $\mathcal{O}(n\sigma + r)$-time reduction to computing uniform σ-runs.

When reporting all p-squares that are distinct as strings, it suffices to replace the $\mathsf{LPF}^{\approx}$ array with the standard $\mathsf{LPF} = \mathsf{LPF}^{=}$ array. Such an array can be computed in $\mathcal{O}(n)$ time after the letters of the string have been sorted [12]. Then the computations take $\mathcal{O}(n\sigma + r + \mathsf{output})$ time. ◀

By Theorem 12, string $\overrightarrow{T}$ (and $\overleftarrow{T}$) contains $\mathcal{O}(n\sigma \log \sigma)$ uniform σ-runs. They can be computed in $\mathcal{O}(n\sigma \log \sigma)$ time; we use the algorithm of Kolpakov and Kucherov [34, 35] to compute all σ-runs in $\overrightarrow{T}$ and then partition each σ-run to uniform σ-runs using kangaroo jumps. That is, let $\overrightarrow{T}[a\,..\,b)$ be a σ-run with period. We compute two values:

$$d_1 = 1 + \mathsf{LCP}(\overrightarrow{T}[a\,..\,b-2\ell), \overrightarrow{T}[a+\ell\,..\,b-\ell))$$
$$d_2 = 1 + \mathsf{LCP}(\overrightarrow{T}[a+\ell\,..\,b-\ell), \overrightarrow{T}[a+2\ell\,..\,b))$$

that, intuitively, find the first ℓ-mismatching position in the σ-run, if any (d_1), and the first ℓ-mismatching position after position $a+\ell-1$, if any (d_2). Let $d = \min(d_1, d_2)$. We then report a uniform σ-run $\overrightarrow{T}[a\,..\,a+d+2\ell)$ and continue processing the (non-maximal) σ-run $\overrightarrow{T}[a+d\,..\,b)$ until its length drops below 2ℓ. The LCP-queries in $(\overrightarrow{T})^R$ are answered in $\mathcal{O}(1)$ time [5, 25]. Thus Lemma 22 implies Theorem 2.

5 Counting Generalised Squares in $\mathcal{O}(n \log n)$ Time

In this section we show an algorithm that counts non-equivalent $\approx$-squares, for an SCER $\approx$ with encoding function $\mathcal{E}$, in $\mathcal{O}(\pi^{\mathcal{E}}(n) + n\rho^{\mathcal{E}}(n) + n \log n)$ time.

For a string T of length n and positive integer $p \leq n/2$, we denote

$$\mathsf{Squares}_p^{\approx} = \{i \in [1\,..\,n-2p+1] \,:\, T[i\,..\,i+2p) \text{ is an } \approx\text{-square}\}.$$

An *interval representation* of a set X of integers is $X = [i_1\,..\,j_1] \cup [i_2\,..\,j_2] \cup \cdots \cup [i_t\,..\,j_t]$, where $j_1 + 1 < i_2, \ldots, j_{t-1} + 1 < i_t$; t is called the *size* of the representation.

Counterparts of the following lemma corresponding to order-preserving squares and Cartesian-tree squares were shown in [21] and [44], respectively. Our proof generalises these proofs; it can be found in the full version.

▶ **Lemma 25.** *Let $\approx$ be an SCER and $\mathcal{E}$ be its encoding function. Given a string T of length n, the interval representations of the sets $\mathsf{Squares}_p^{\approx}$ for all $1 \leq p \leq n/2$ have total size $\mathcal{O}(n \log n)$ and can be computed in $\mathcal{O}(\pi^{\mathcal{E}}(n) + n\rho^{\mathcal{E}}(n) + n \log n)$ time.*

We count non-equivalent $\approx$-squares using the $\mathsf{LPF}^{\approx}$ array. We would like to count an $\approx$-square at the position of its leftmost occurrence. For an integer array $A[1\,..\,n]$, we denote a range count query for $i, j \in [1\,..\,n]$, $x \in \mathbb{Z}$ as $\mathsf{RangeCount}_A(i, j, x) = |\{k \in [i\,..\,j] \,:\, A[k] < x\}|$.

Then our problem reduces to computing the sum

$$\sum_{p=1}^{\lfloor n/2 \rfloor} \sum_{[i\,.\,.\,j] \in \mathsf{Squares}_p} \mathsf{RangeCount}_{\mathsf{LPF}^{\approx}}(i, j, 2p). \tag{4}$$

We say that an array $A[1\,.\,.\,n]$ is a *linear oscillation array* if $\sum_{i=1}^{n-1} |A[i+1] - A[i]| = \mathcal{O}(n)$. The following fact is folklore for the LPF array; we prove it for $\mathsf{LPF}^{\approx}$ in the full version.

▶ **Lemma 26.** *For any SCER $\approx$, $\mathsf{LPF}^{\approx}$ is a linear oscillation array.*

We will now show that the sum from Equation (4) can be computed in $\mathcal{O}(n \log n)$ time. We use a very simple abstract problem.

Counting Problem
Input: A set $Y \subseteq [1\,.\,.\,n]$, initially empty, and an integer ℓ, initially $\ell = 0$.
Operation: One of: (1) insert an element $x \in [1\,.\,.\,n]$ to Z; (2) delete an element $x \in Y$ from Y; (3) increment ℓ by 1; (4) decrement ℓ by 1. After each operation, output $|Y \cap [\ell+1\,.\,.\,n]|$.

The Counting Problem can be solved with a basic indicator array for Y.

▶ **Lemma 27.** *After $\mathcal{O}(n)$-time preprocessing, each operation in the Counting Problem can be answered in $\mathcal{O}(1)$ time.*

Proof. We store an indicator array of bits $C[1\,.\,.\,n]$ such that $C[i] = 1$ if and only if $i \in Y$. Initially, $C \equiv 0$. We also store a value $a = |Y \cap [\ell + 1\,.\,.\,n]|$; initially $a = 0$. After each operation, the current value of a is returned.

When we insert an element $x \in [1\,.\,.\,n]$ to Y, we set $C[x]$ to 1 and increment a if $x > \ell$. Symmetrically, when we delete $x \in Y$ from Y, we set $C[x]$ to 0 and decrement a if $x < \ell$. When we increment ℓ, we subtract $C[\ell]$ from a. When we decrement ℓ, we add $C[\ell]$ to a. ◀

We are ready to obtain the final main result.

Proof of Theorem 5. First we show how to compute the number of non-equivalent $\approx$-squares in a length-n string. The $\mathsf{LPF}^{\approx}$ array can be computed in $\mathcal{O}(\pi^{\mathcal{E}}(n) + n\rho^{\mathcal{E}}(n) + n \log^2 \log n / \log\log\log n)$ time (Corollary 15). We compute the interval representations of the sets $\mathsf{Squares}_p^{\approx}$ using Lemma 25 in $\mathcal{O}(\pi^{\mathcal{E}}(n) + n\rho^{\mathcal{E}}(n) + n \log n)$ time. The interval representations have total size $\mathcal{O}(n \log n)$. We need to compute the sum of results of $\mathcal{O}(n \log n)$ $\mathsf{RangeCount}$ queries as in Equation (4). We will do it using the Counting Problem.

Let us bucket sort all start and end points of intervals across all sets $\mathsf{Squares}_p^{\approx}$ in $\mathcal{O}(n \log n)$ total time. We create an instance of the Counting Problem. For each position k from 1 to n, we proceed as follows. First, for each interval $[k\,.\,.\,j] \in \mathsf{Squares}_p^{\approx}$, for any p, we insert $2p$ to the set Y. Then, we change the current value ℓ in the problem to $\mathsf{LPF}^{\approx}[k]$ by performing increments or decrements, as appropriate. Afterwards, we add the returned value of the Counting Problem to the final result. Finally, for each interval $[i\,.\,.\,k] \in \mathsf{Squares}_p^{\approx}$, for any p, we delete $2p$ from the set Y.

For each $p \in [1\,.\,.\,\lfloor n/2 \rfloor]$, the intervals in $\mathsf{Squares}_p$ are pairwise disjoint. By Lemma 25, $\mathcal{O}(n \log n)$ insertions and deletions are performed in the Counting Problem. The total number of increments and decrements is bounded by $\mathcal{O}(n)$ by Lemma 26 (and the fact that the values in this array are in $[0\,.\,.\,n)$). In total, the sum (4) is computed in $\mathcal{O}(n \log n)$ time. We obtain the first part of Theorem 5.

As before, if one wishes to compute all $\approx$-squares that are distinct as substrings, it suffices to replace the $\mathsf{LPF}^{\approx}$ array in the algorithm by the standard $\mathsf{LPF} = \mathsf{LPF}^{=}$ array. We obtain the second part of Theorem 5. ◀

References

1 Arne Andersson and Mikkel Thorup. Dynamic ordered sets with exponential search trees. *Journal of the ACM*, 54(3):13, 2007. doi:10.1145/1236457.1236460.

2 Brenda S. Baker. Parameterized pattern matching: Algorithms and applications. *Journal of Computer and System Sciences*, 52(1):28–42, 1996. doi:10.1006/JCSS.1996.0003.

3 Hideo Bannai, Tomohiro I, Shunsuke Inenaga, Yuto Nakashima, Masayuki Takeda, and Kazuya Tsuruta. The "Runs" theorem. *SIAM Journal on Computing*, 46(5):1501–1514, 2017. doi:10.1137/15M1011032.

4 Hideo Bannai, Shunsuke Inenaga, and Dominik Köppl. Computing all distinct squares in linear time for integer alphabets. In Juha Kärkkäinen, Jakub Radoszewski, and Wojciech Rytter, editors, *28th Annual Symposium on Combinatorial Pattern Matching, CPM 2017, July 4-6, 2017, Warsaw, Poland*, volume 78 of *LIPIcs*, pages 22:1–22:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPICS.CPM.2017.22.

5 Michael A. Bender and Martin Farach-Colton. The LCA problem revisited. In Gaston H. Gonnet, Daniel Panario, and Alfredo Viola, editors, *LATIN 2000: Theoretical Informatics, 4th Latin American Symposium, Punta del Este, Uruguay, April 10-14, 2000, Proceedings*, volume 1776 of *Lecture Notes in Computer Science*, pages 88–94. Springer, 2000. doi:10.1007/10719839_9.

6 Gary Benson. An algorithm for finding tandem repeats of unspecified pattern size. In Sorin Istrail, Pavel A. Pevzner, and Michael S. Waterman, editors, *Proceedings of the Second Annual International Conference on Research in Computational Molecular Biology, RECOMB 1998, New York, NY, USA, March 22-25, 1998*, pages 20–29. ACM, 1998. doi:10.1145/279069.279079.

7 Gary Benson. Tandem repeats finder: a program to analyze DNA sequences. *Nucleic Acids Research*, 27(2):573–580, 1999. doi:10.1093/nar/27.2.573.

8 Srečko Brlek and Shuo Li. On the number of squares in a finite word. *Combinatorial Theory*, 5(1), 2025. doi:10.5070/C65165014.

9 Panagiotis Charalampopoulos, Tomasz Kociumaka, Jakub Radoszewski, Wojciech Rytter, Tomasz Waleń, and Wiktor Zuba. Efficient enumeration of distinct factors using package representations. In Christina Boucher and Sharma V. Thankachan, editors, *String Processing and Information Retrieval – 27th International Symposium, SPIRE 2020, Orlando, FL, USA, October 13-15, 2020, Proceedings*, volume 12303 of *Lecture Notes in Computer Science*, pages 247–261. Springer, 2020. doi:10.1007/978-3-030-59212-7_18.

10 Panagiotis Charalampopoulos, Manal Mohamed, Jakub Radoszewski, Wojciech Rytter, Tomasz Waleń, and Wiktor Zuba. Counting distinct square substrings in sublinear time. In Paweł Gawrychowski, Filip Mazowiecki, and Michał Skrzypczak, editors, *50th International Symposium on Mathematical Foundations of Computer Science, MFCS 2025, August 25–29, 2025, Warsaw, Poland*, LIPIcs, 2025. doi:10.4230/LIPICS.MFCS.2025.36.

11 Richard Cole and Ramesh Hariharan. Faster suffix tree construction with missing suffix links. *SIAM Journal on Computing*, 33(1):26–42, 2003. doi:10.1137/S0097539701424465.

12 Maxime Crochemore and Lucian Ilie. Computing longest previous factor in linear time and applications. *Information Processing Letters*, 106(2):75–80, 2008. doi:10.1016/J.IPL.2007.10.006.

13 Maxime Crochemore, Costas S. Iliopoulos, Tomasz Kociumaka, Marcin Kubica, Alessio Langiu, Solon P. Pissis, Jakub Radoszewski, Wojciech Rytter, and Tomasz Waleń. Order-preserving indexing. *Theoretical Computer Science*, 638:122–135, 2016. doi:10.1016/J.TCS.2015.06.050.

14 Maxime Crochemore, Costas S. Iliopoulos, Marcin Kubica, Jakub Radoszewski, Wojciech Rytter, and Tomasz Waleń. Extracting powers and periods in a word from its runs structure. *Theoretical Computer Science*, 521:29–41, 2014. doi:10.1016/J.TCS.2013.11.018.

15 Olivier Delgrange and Eric Rivals. STAR: an algorithm to search for tandem approximate repeats. *Bioinformatics*, 20:2812–2820, 2004. doi:10.1093/bioinformatics/bth335.

16 Nevzat Onur Domaniç and Franco P. Preparata. A novel approach to the detection of genomic approximate tandem repeats in the Levenshtein metric. *Journal of Computational Biology*, 14(7):873–891, 2007. PMID: 17803368. doi:10.1089/cmb.2007.0018.

17 Jonas Ellert and Johannes Fischer. Linear time runs over general ordered alphabets. In Nikhil Bansal, Emanuela Merelli, and James Worrell, editors, *48th International Colloquium on Automata, Languages, and Programming, ICALP 2021, July 12-16, 2021, Glasgow, Scotland (Virtual Conference)*, volume 198 of *LIPIcs*, pages 63:1–63:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICS.ICALP.2021.63.

18 Younan Gao, Meng He, and Yakov Nekrich. Fast preprocessing for optimal orthogonal range reporting and range successor with applications to text indexing. In Fabrizio Grandoni, Grzegorz Herman, and Peter Sanders, editors, *28th Annual European Symposium on Algorithms, ESA 2020, September 7-9, 2020, Pisa, Italy (Virtual Conference)*, volume 173 of *LIPIcs*, pages 54:1–54:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPICS.ESA.2020.54.

19 Pawel Gawrychowski, Samah Ghazawi, and Gad M. Landau. Order-preserving squares in strings. In Laurent Bulteau and Zsuzsanna Lipták, editors, *34th Annual Symposium on Combinatorial Pattern Matching, CPM 2023, June 26-28, 2023, Marne-la-Vallée, France*, volume 259 of *LIPIcs*, pages 13:1–13:19. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICS.CPM.2023.13.

20 Pawel Gawrychowski, Tomohiro I, Shunsuke Inenaga, Dominik Köppl, and Florin Manea. Tighter bounds and optimal algorithms for all maximal α-gapped repeats and palindromes – finding all maximal α-gapped repeats and palindromes in optimal worst case time on integer alphabets. *Theory of Computing Systems*, 62(1):162–191, 2018. doi:10.1007/S00224-017-9794-5.

21 Garance Gourdel, Tomasz Kociumaka, Jakub Radoszewski, Wojciech Rytter, Arseny M. Shur, and Tomasz Waleń. String periods in the order-preserving model. *Information and Computation*, 270, 2020. doi:10.1016/J.IC.2019.104463.

22 Dan Gusfield. *Algorithms on Strings, Trees, and Sequences – Computer Science and Computational Biology*. Cambridge University Press, 1997. doi:10.1017/CBO9780511574931.

23 Dan Gusfield and Jens Stoye. Linear time algorithms for finding and representing all the tandem repeats in a string. *Journal of Computer and System Sciences*, 69(4):525–546, 2004. doi:10.1016/J.JCSS.2004.03.004.

24 Rikuya Hamai, Kazushi Taketsugu, Yuto Nakashima, Shunsuke Inenaga, and Hideo Bannai. On the number of non-equivalent parameterized squares in a string. In Zsuzsanna Lipták, Edleno Silva de Moura, Karina Figueroa, and Ricardo Baeza-Yates, editors, *String Processing and Information Retrieval – 31st International Symposium, SPIRE 2024, Puerto Vallarta, Mexico, September 23-25, 2024, Proceedings*, volume 14899 of *Lecture Notes in Computer Science*, pages 174–183. Springer, 2024. doi:10.1007/978-3-031-72200-4_13.

25 Dov Harel and Robert Endre Tarjan. Fast algorithms for finding nearest common ancestors. *SIAM Journal on Computing*, 13(2):338–355, 1984. doi:10.1137/0213024.

26 Tomohiro I, Shunsuke Inenaga, and Masayuki Takeda. Palindrome pattern matching. *Theoretical Computer Science*, 483:162–170, 2013. doi:10.1016/J.TCS.2012.01.047.

27 Tomohiro I and Dominik Köppl. Improved upper bounds on all maximal α-gapped repeats and palindromes. *Theoretical Computer Science*, 753:1–15, 2019. doi:10.1016/J.TCS.2018.06.033.

28 Haim Kaplan, Ely Porat, and Nira Shafrir. Finding the position of the k-mismatch and approximate tandem repeats. In Lars Arge and Rusins Freivalds, editors, *Algorithm Theory – SWAT 2006, 10th ScandinavianWorkshop on Algorithm Theory, Riga, Latvia, July 6-8, 2006, Proceedings*, volume 4059 of *Lecture Notes in Computer Science*, pages 90–101. Springer, 2006. doi:10.1007/11785293_11.

29 Jinil Kim, Peter Eades, Rudolf Fleischer, Seok-Hee Hong, Costas S. Iliopoulos, Kunsoo Park, Simon J. Puglisi, and Takeshi Tokuyama. Order-preserving matching. *Theoretical Computer Science*, 525:68–79, 2014. doi:10.1016/J.TCS.2013.10.006.

30 Tomasz Kociumaka, Jakub Radoszewski, Wojciech Rytter, and Tomasz Waleń. Maximum number of distinct and nonequivalent nonstandard squares in a word. *Theoretical Computer Science*, 648:84–95, 2016. doi:10.1016/J.TCS.2016.08.010.

31 Roman Kolpakov, Ghizlane Bana, and Gregory Kucherov. mreps: Efficient and flexible detection of tandem repeats in DNA. *Nucleic Acids Research*, 31:3672–3678, 2003. doi:10.1093/nar/gkg617.

32 Roman Kolpakov, Mikhail Podolskiy, Mikhail Posypkin, and Nickolay Khrapov. Searching of gapped repeats and subrepetitions in a word. *Journal of Discrete Algorithms*, 46-47:1–15, 2017. doi:10.1016/J.JDA.2017.10.004.

33 Roman M. Kolpakov and Gregory Kucherov. Finding maximal repetitions in a word in linear time. In *40th Annual Symposium on Foundations of Computer Science, FOCS 1999*, pages 596–604. IEEE Computer Society, 1999. doi:10.1109/SFFCS.1999.814634.

34 Roman M. Kolpakov and Gregory Kucherov. Finding approximate repetitions under Hamming distance. In Friedhelm Meyer auf der Heide, editor, *Algorithms – ESA 2001, 9th Annual European Symposium, Aarhus, Denmark, August 28-31, 2001, Proceedings*, volume 2161 of *Lecture Notes in Computer Science*, pages 170–181. Springer, 2001. doi:10.1007/3-540-44676-1_14.

35 Roman M. Kolpakov and Gregory Kucherov. Finding approximate repetitions under Hamming distance. *Theoretical Computer Science*, 303(1):135–156, 2003. doi:10.1016/S0304-3975(02)00448-6.

36 Arun Krishnan and Francis Tang. Exhaustive whole-genome tandem repeats search. *Bioinformatics*, 20(16):2702–2710, May 2004. doi:10.1093/bioinformatics/bth311.

37 Marcin Kubica, Tomasz Kulczyński, Jakub Radoszewski, Wojciech Rytter, and Tomasz Waleń. A linear time algorithm for consecutive permutation pattern matching. *Information Processing Letters*, 113(12):430–433, 2013. doi:10.1016/J.IPL.2013.03.015.

38 Gregory Kucherov and Dina Sokol. Approximate tandem repeats. In Ming-Yang Kao, editor, *Encyclopedia of Algorithms*, pages 1–6. Springer Berlin Heidelberg, Berlin, Heidelberg, 2014. doi:10.1007/978-3-642-27848-8_24-2.

39 Gad M. Landau, Jeanette P. Schmidt, and Dina Sokol. An algorithm for approximate tandem repeats. *Journal of Computational Biology*, 8(1):1–18, 2001. doi:10.1089/106652701300099038.

40 Michael G. Main and Richard J. Lorentz. An O(n log n) algorithm for finding all repetitions in a string. *Journal of Algorithms*, 5(3):422–432, 1984. doi:10.1016/0196-6774(84)90021-X.

41 Glenn K. Manacher. A new linear-time "on-line" algorithm for finding the smallest initial palindrome of a string. *Journal of the ACM*, 22(3):346–351, 1975. doi:10.1145/321892.321896.

42 Yoshiaki Matsuoka, Takahiro Aoki, Shunsuke Inenaga, Hideo Bannai, and Masayuki Takeda. Generalized pattern matching and periodicity under substring consistent equivalence relations. *Theoretical Computer Science*, 656:225–233, 2016. doi:10.1016/J.TCS.2016.02.017.

43 Angelika Merkel and Neal Gemmell. Detecting short tandem repeats from genome data: opening the software black box. *Briefings in Bioinformatics*, 9:355–366, 2008. doi:10.1093/bib/bbn028.

44 Sung Gwan Park, Magsarjav Bataa, Amihood Amir, Gad M. Landau, and Kunsoo Park. Finding patterns and periods in Cartesian tree matching. *Theoretical Computer Science*, 845:181–197, 2020. doi:10.1016/J.TCS.2020.09.014.

45 Marco Pellegrini, M. Elena Renda, and Alessio Vecchio. TRStalker: an efficient heuristic for finding fuzzy tandem repeats. *Bioinformatics*, 26(12):i358–i366, June 2010. doi:10.1093/bioinformatics/btq209.

46 Jeff Reneker, Chi-Ren Shyu, Peiyu Zeng, Joseph C. Polacco, and Walter Gassmann. ACMES: fast multiple-genome searches for short repeat sequences with concurrent cross-species information retrieval. *Nucleic Acids Research*, 32:W649–W653, 2004. doi:10.1093/nar/gkh455.

47 E. Rivals, J. Delahaye, M. Dauchet, and O. Delgrange. A first step toward chromosome analysis by compression algorithms. In *Proceedings First International Symposium on Intelligence in Neural and Biological Systems. INBS 1995*, pages 233–239, 1995. doi:10.1109/INBS.1995.404256.

48 Eric Rivals, Olivier Delgrange, Jean-Paul Delahaye, Max Dauchet, Marie-Odile Delorme, Alain Hénaut, and Emmanuelle Ollivier. Detection of significant patterns by compression algorithms: the case of approximate tandem repeats in DNA sequences. *Computer Applications in the Biosciences*, 13(2):131–136, 1997. doi:10.1093/BIOINFORMATICS/13.2.131.

49 Marie-France Sagot and Eugene W. Myers. Identifying satellites and periodic repetitions in biological sequences. *Journal of Computational Biology*, 5(3):539–553, 1998. doi:10.1089/CMB.1998.5.539.

50 Tiago José P. Sobreira, Alan M. Durham, and Arthur Gruber. TRAP: automated classification, quantification and annotation of tandemly repeated sequences. *Bioinformatics*, 22(3):361–362, 2006. doi:10.1093/bioinformatics/bti809.

51 Dina Sokol, Gary Benson, and Justin Tojeira. Tandem repeats over the edit distance. *Bioinformatics*, 23(2):e30–e35, January 2007. doi:10.1093/bioinformatics/btl309.

52 Dina Sokol and Justin Tojeira. Speeding up the detection of tandem repeats over the edit distance. *Theoretical Computer Science*, 525:103–110, 2014. Advances in Stringology. doi:10.1016/j.tcs.2013.04.021.

53 Axel Thue. Über unendliche Zeichenreihen. *Norske Videnskabers Selskabs Skrifter Mat.-Nat. Kl.*, 7:1–22, 1906.

54 Jean Vuillemin. A unifying look at data structures. *Communications of the ACM*, 23(4):229–239, April 1980. doi:10.1145/358841.358852.

55 Ydo Wexler, Zohar Yakhini, Yechezkel Kashi, and Dan Geiger. Finding approximate tandem repeats in genomic sequences. In Philip E. Bourne and Dan Gusfield, editors, *Proceedings of the Eighth Annual International Conference on Computational Molecular Biology, 2004, San Diego, California, USA, March 27-31, 2004*, pages 223–232. ACM, 2004. doi:10.1145/974614.974644.

56 Hongxia Zhou, Liping Du, and Hong Yan. Detection of tandem repeats in dna sequences based on parametric spectral estimation. *IEEE Transactions on Information Technology in Biomedicine*, 13(5):747–755, 2009. doi:10.1109/TITB.2008.920626.