Algorithmica (2022) 84:566–589
https://doi.org/10.1007/s00453-021-00842-7

# *k*-Approximate Quasiperiodicity Under Hamming and Edit Distance

Aleksander Kędzierski[1,2] · Jakub Radoszewski[1,2]

Received: 1 September 2020 / Accepted: 5 June 2021 / Published online: 22 June 2021

**Abstract**
Quasiperiodicity in strings was introduced almost 30 years ago as an extension of string periodicity. The basic notions of quasiperiodicity are cover and seed. A cover of a text $T$ is a string whose occurrences in $T$ cover all positions of $T$. A seed of text $T$ is a cover of a superstring of $T$. In various applications exact quasiperiodicity is still not sufficient due to the presence of errors. We consider approximate notions of quasiperiodicity, for which we allow approximate occurrences in $T$ with a small Hamming, Levenshtein or weighted edit distance. In previous work Sim et al. (J Korea Inf Sci Soc 29(1):16–21, 2002) and Christodoulakis et al. (J Autom Lang Comb 10(5/6), 609–626, 2005) showed that computing approximate covers and seeds, respectively, under weighted edit distance is NP-hard. They, therefore, considered restricted approximate covers and seeds which need to be factors of the original string $T$ and presented polynomial-time algorithms for computing them. Further algorithms, considering approximate occurrences with Hamming distance bounded by $k$, were given in several contributions by Guth et al. They also studied relaxed approximate quasiperiods. We present more efficient algorithms for computing restricted approximate covers and seeds. In particular, we improve upon the complexities of many of the aforementioned algorithms, also for relaxed quasiperiods. Our solutions are especially efficient if the number (or total cost) of allowed errors is small. We also show conditional lower bounds for computing restricted approximate covers and prove NP-hardness of computing non-restricted approximate covers and seeds under the Hamming distance.

**Keywords** Quasiperiodicity · Approximate cover · Approximate seed · Hamming distance · Edit distance

✉ Jakub Radoszewski
jrad@mimuw.edu.pl

Aleksander Kędzierski
akedzierski@mimuw.edu.pl

[1] Institute of Informatics, University of Warsaw, Warsaw, Poland

[2] Samsung R&D Institute Poland, Warsaw, Poland

## 1 Introduction

Quasiperiodicity was introduced as an extension of periodicity [4]. Its aim is to capture repetitive structure of strings that do not have an exact period. The basic notions of quasiperiodicity are cover (also called quasiperiod) and seed. A *cover* of a string $T$ is a string $C$ whose occurrences cover all positions of $T$. A *seed* of string $T$ is a cover of a superstring of $T$. Covers and seeds were first considered in [5, 19], respectively, and linear-time algorithms computing them are known; see [6, 19, 25, 30–32].

A cover is necessarily a *border*, that is, a prefix and a suffix of the string. A seed $C$ of $T$ covers all positions of $T$ by its occurrences or by left- or right-overhangs, that is, by suffixes of $C$ being prefixes of $T$ and prefixes of $C$ being suffixes of $T$. In order to avoid extreme cases one usually assumes that covers $C$ of $T$ need to satisfy $|C| < |T|$ and seeds $C$ need to satisfy $2|C| \leq |T|$ (so a seed needs to be a factor of $T$). Seeds, unlike covers, preserve an important property of periods that if $T$ has a period or a seed, then every (sufficiently long) factor of $T$ has the same period or seed, respectively.

The classic notions of quasiperiodicity may not capture repetitive structure of strings in practical settings; it was also confirmed by a recent experimental study [10]. In order to tackle this problem, further types of quasiperiodicity were studied that require that only a certain number of positions in a string are covered. This way notions of enhanced cover, partial cover and partial seed were introduced. A *partial cover* and *partial seed* are required to cover a given number of positions of a string, where overhangs are allowed for the partial seed, and an *enhanced cover* is a partial cover with an additional requirement of being a border of the string. $\mathcal{O}(n \log n)$-time algorithms for computing shortest partial covers and seeds were shown in [26, 27], respectively, whereas a linear-time algorithm for computing a proper enhanced cover that covers the maximum number of positions in $T$ was presented (among other variations of the problem) in [12].

Further study has lead to *approximate quasiperiodicity* in which approximate occurrences of a quasiperiod are allowed. In particular, Hamming, Levenshtein and weighted edit distance were considered. A *k-approximate cover* of string $T$ is a string $C$ whose approximate occurrences with distance at most $k$ cover $T$; see Fig. 1. Similarly one can define a *k-approximate seed*, allowing overhangs. These notions were introduced by Sim et al. [34] and Christodoulakis et al. [8], respectively, who showed that the problem of checking if a string $T$ has a $k$-approximate cover and $k$-approximate seed, respectively, for a given $k$ is NP-complete under weighted edit distance. (Their proof used arbitrary integer weights and a constant-sized—12 letters in the case of approximate seeds—alphabet.) Therefore, they considered a *restricted* version of the problem in which the approximate cover or seed is required to be a

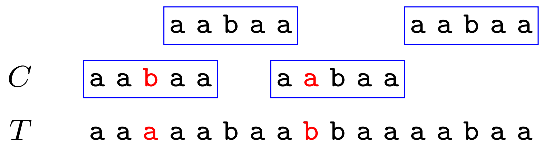

**Fig. 1** String $C =$ aabaa is a (restricted) 1-approximate cover of the text $T$ under the Hamming distance. Mismatches between $C$ and its approximate occurrences in $T$ are highlighted

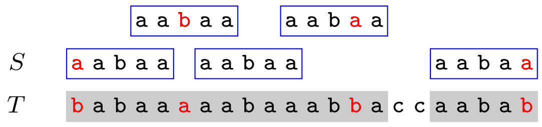

**Fig. 2** The 1-coverage of string $S =$ aabaa in text $T$ under the Hamming distance is 20

factor of $T$. Formally, the problem is to compute, for every factor of $T$, the smallest $k$ for which it is a $k$-approximate cover or seed of $T$. For this version of the problem, they presented an $\mathcal{O}(n^3)$-time algorithm for the Hamming distance and an $\mathcal{O}(n^4)$-time algorithm for the edit distance[1]. The same problems under Hamming distance were considered by Guth et al. [17] and Guth and Melichar [16]. They studied a *k-restricted* version of the problems, in which we are only interested in factors of $T$ being $\ell$-approximate covers or seeds for $\ell \le k$, and developed $\mathcal{O}(n^3(|\Sigma| + k))$-time and $\mathcal{O}(n^3|\Sigma|k)$-time automata-based algorithms for $k$-restricted approximate covers and seeds, respectively. Experimental evaluation of these algorithms was performed by Guth [14].

Recently, Guth [15] extended this study to *k-approximate restricted enhanced covers* under Hamming distance. In this problem, we search for a border of $T$ whose $k$-approximate occurrences cover the maximum number of text positions. In another variant of the problem, which one could see as approximate partial cover problem, we only require the approximate enhanced cover to be a $k$-approximate border of $T$, but still to be a factor of $T$. Guth [15] proposed $\mathcal{O}(n^2)$-time and $\mathcal{O}(n^3(|\Sigma| + k))$-time algorithms for the two respective variants.

We improve upon previous results on restricted approximate quasiperiodicity. We introduce a general notion of *k-coverage* of a string $S$ in a string $T$, defined as the number of positions in $T$ that are covered by $k$-approximate occurrences of $S$; see Fig. 2. Efficient algorithms computing the $k$-coverage for factors of $T$ are presented. We also show NP-hardness for non-restricted approximate covers and seeds under the Hamming distance. A detailed list of our results is as follows.

1. The Hamming $k$-coverage for every prefix and for every factor of a string of length $n$ can be computed in (A) $\mathcal{O}(n + \min\{\, nk, nk^{2/3}\log^{1/3} n \log k \,\})$ time (for a string over an integer alphabet) and in the optimal (B) $\mathcal{O}(n^2)$ time, respectively. (See Sect. 3.)

   With this result we obtain algorithms with the time complexities (A) and (B) for the two versions of $k$-approximate restricted enhanced covers that were proposed by Guth [15] and an $\mathcal{O}(n^2k)$-time algorithm computing $k$-restricted approximate covers and seeds. Our algorithms work in the optimal $\mathcal{O}(n)$ and $\mathcal{O}(n^2)$ time, respectively, when computing 0-coverage of prefixes and factors. Moreover, our algorithm for prefixes actually works in linear time assuming that a $k$-mismatch version of the PREF table [9] is given. Thus, as a by-product, for $k = 0$, we obtain

[1] In fact, they consider *relative* Hamming and Levenshtein distances which are inversely proportional to the length of the candidate factor and seek for an approximate cover/seed that minimizes such distance. However, their algorithms actually compute the minimum distance $k$ for every factor of $T$ under the standard distance definitions.

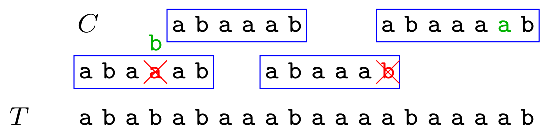

**Fig. 3** String $C =$ abaaab is a (restricted) 1-approximate cover of the text $T$ under the Levenshtein distance. Example occurrences with a substitution, a deletion and an insertion are shown

an alternative linear-time algorithm for computing all (exact) enhanced covers of a string. (A different linear-time algorithm for this problem was given in [12]).

The complexities come from using tools of Kaplan et al. [22] and Flouri et al. [11], respectively.

2. The $k$-coverage under Levenshtein distance and weighted edit distance for every factor of a string of length $n$ can be computed in $\mathcal{O}(n^3)$ time and $\mathcal{O}(n^3\sqrt{n\log n})$ time, respectively. (See Sect. 4.)

   We also show in Sect. 4 how our approach can be used to compute restricted approximate covers and seeds under weighted edit distance in $\mathcal{O}(n^3\sqrt{n\log n})$ time, thus improving upon the previous $\mathcal{O}(n^4)$-time algorithms of Sim et al. [34] and Christodoulakis et al. [8]. (See also Fig. 3.)

   Our algorithm for Levenshtein distance uses incremental string comparison [28].

3. Under Hamming distance, it is NP-hard to check if a given string of length $n$ has a $k$-approximate cover or a $k$-approximate seed of a given length $c$. Moreover, restricted approximate covers of a given length cannot be computed in $\mathcal{O}(n^{2-\varepsilon})$ time, for $\varepsilon > 0$, unless the Strong Exponential Time Hypothesis (SETH) of Impagliazzo, Paturi, and Zane [20, 21] is false. These statements hold even for strings over a binary alphabet. (See Sect. 5.)

   The first result extends the previous proofs of Sim et al. [34] and Christodoulakis et al. [8] which worked for the weighted edit distance. The conditional lower bound is based on conditional hardness of the Orthogonal Vectors problem that was introduced by Williams [36].

A different notion of approximate cover, which we do not consider in this work, was recently studied in [1–3], where it was assumed that the string $T$ may not have a cover, but it is at a small Hamming distance from a string $T'$ that has a proper cover. These works defined an approximate cover of $T$ as the shortest cover of a string $T'$ that is closest to $T$ under Hamming distance. Interestingly, this problem was also shown to be NP-hard [1] and an $\mathcal{O}(n^4)$-time algorithm was developed for it in the restricted case that the approximate cover is a factor of the string $T$ [2].

Let us note that under this definition the total number of substitutions is counted, instead of the maximum number of mismatches as in our study. Moreover, it is not true that if $T$ has a $k$-approximate cover under our definition that has length $\ell$ and $t$ occurrences in $T$, then there is a string $T'$ that is at Hamming distance $kt$ to $T$ and has a cover of length $\ell$. This is because, under our definition, one letter of $T$ can correspond to two different letters in occurrences of an approximate cover. E.g., $T =$ acb has a 1-approximate cover ab under our definition, but there is no string at Hamming distance at most 2 from $T$ that would have a proper cover. Our work can be viewed as complementary to the study of [1–3] as "the natural definition of an approximate repetition is not clear" [2].

This is a full version of a conference paper [23]. In particular, the conference version is extended by the conditional lower bound.

## 2 Preliminaries

We consider strings over an alphabet $\Sigma$. The empty string is denoted by $\varepsilon$. For a string $T$, by $|T|$ we denote its length and by $T[0], \dots, T[|T|-1]$ its subsequent letters. By $T[i,j]$ we denote the string $T[i] \dots T[j]$ which we call a *factor* of $T$. If $i = 0$, it is a *prefix* of $T$, and if $j = |T| - 1$, it is a *suffix* of $T$. A string that is both a prefix and a suffix of $T$ is called a *border* of $T$. For a string $T = XY$ such that $|X| = b$, by $\mathsf{rot}_b(T)$ we denote $YX$, called a *cyclic shift* of $T$.

For equal-length strings $U$ and $V$, by $Ham(U, V)$ we denote their *Hamming distance*, that is, the number of positions where they do not match. For strings $U$ and $V$, by $ed(U, V)$ we denote their *edit distance*, that is, the minimum cost of edit operations (insertions, deletions, substitutions) that allow to transform $U$ to $V$. Here the cost of an edit operation can vary depending both on the type of the operation and on the letters that take part in it. In case that all edit operations have unit cost, the edit distance is also called *Levenshtein distance* and denoted here as $Lev(U, V)$.

For two strings $S$ and $T$ and metric $d$, we denote by

$$\mathsf{Occ}^d_k(S,T) = \{[i,j] \ : \ d(S, T[i,j]) \le k\}$$

the set of approximate occurrences of $S$ in $T$, represented as intervals, under the metric $d$. We then denote by

$$\mathsf{Covered}^d_k(S,T) = |\bigcup \mathsf{Occ}^d_k(S,T)|$$

the *k-coverage* of $S$ in $T$. In case of Hamming or Levenshtein distances we can assume that $k \le n$, but for the weighted edit distance $k$ can be arbitrarily large. Moreover, by $\mathsf{StartOcc}^d_k(S,T)$ we denote the set of left endpoints of the intervals in $\mathsf{Occ}^d_k(S,T)$.

**Definition 1** Let $d$ be a metric and $T$ be a string. We say that string $C$, $|C| < |T|$, is a *k-approximate cover* of $T$ under metric $d$ if $\mathsf{Covered}^d_k(C,T) = |T|$.

We say that string $C$, $2|C| \le |T|$, is a *k-approximate seed* of $T$ if it is a $k$-approximate cover of some string $T'$ whose factor is $T$. Let $\diamondsuit$ be a wildcard symbol that matches every other symbol of the alphabet. Strings over $\Sigma \cup \{\diamondsuit\}$ are also called *partial words*. In order to compute $k$-approximate seeds, it suffices to consider $k$-approximate covers of $\diamondsuit^{|T|}T\diamondsuit^{|T|}$.

The main problems in scope can now be stated as follows.

**Table 1** Example of $\mathsf{PREF}_0$ and $\mathsf{PREF}_1$ arrays

| $i$ | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $T[i]$ | a | b | a | a | a | b | a | b | b | a | a | b | a | b | a | b | a | a | a | a | a | a | b |
| $\mathsf{PREF}_0[i]$ | 23 | 0 | 1 | 1 | 3 | 0 | 2 | 0 | 0 | 1 | 3 | 0 | 3 | 0 | 5 | 0 | 1 | 1 | 1 | 1 | 1 | 2 | 0 |
| $\mathsf{PREF}_1[i]$ | 23 | 1 | 3 | 2 | 4 | 1 | 8 | 4 | 1 | 2 | 7 | 1 | 5 | 1 | 7 | 1 | 5 | 6 | 4 | 3 | 2 | 2 | 1 |

General $k$ -Approximate Cover/Seed

**Input:** String $T$ of length $n$, metric $d$, integer $c \in \{1, \dots, n-1\}$ and number $k$

**Output:** A string $C$ of length $c$ that is a $k$-approximate cover/seed of $T$ under $d$

$k$ -Coverage for Prefixes/Factors

**Input:** String $T$ of length $n$, metric $d$ and number $k$

**Output:** For every prefix/factor of $T$, compute its $k$-coverage under $d$

Restricted Approximate Covers/Seeds

**Input:** String $T$ of length $n$ and metric $d$

**Output:** Compute, for every factor $C$ of $T$, the smallest $k$ such that $C$ is a $k$-approximate cover/seed of $T$ under $d$

## 2.1 Algorithmic Toolbox for Hamming Distance

For a string $T$ of length $n$, by $\mathsf{lcp}_k(i,j)$ we denote the length of the longest common prefix with at most $k$ mismatches of the suffixes $T[i, n-1]$ and $T[j, n-1]$. Flouri et al. [11] proposed an $\mathcal{O}(n^2)$-time algorithm to compute the longest common factor of two strings $T_1$, $T_2$ with at most $k$ mismatches. Their algorithm actually computes the lengths of the longest common prefixes with at most $k$ mismatches of every two suffixes $T_1[i, |T_1|-1]$ and $T_2[j, |T_2|-1]$ and returns the maximum among them. Applied for $T_1 = T_2$, it gives the following result.

**Lemma 1** ([11]) *For a string of length n and an integer* $0 \le k \le n$, *values* $\mathsf{lcp}_k(i,j)$ *for all* $i, j = 0, \dots, n-1$ *can be computed in* $\mathcal{O}(n^2)$ *time.*

We also use a table $\mathsf{PREF}_k$ such that $\mathsf{PREF}_k[i] = \mathsf{lcp}_k(0, i)$; see Table 1. LCP-queries with mismatches can be answered in $\mathcal{O}(k)$ time after linear-time preprocessing using the kangaroo method [29]. In particular, this allows to compute the $\mathsf{PREF}_k$ table in $\mathcal{O}(n + nk)$ time. Kaplan et al. [22] presented an algorithm that, given a pattern $P$ of length $m$, a text $T$ of length $n$ over an integer alphabet $\Sigma \subseteq \{1, \dots, n^{\mathcal{O}(1)}\}$, and an integer $k > 0$, finds in $\mathcal{O}(nk^{2/3} \log^{1/3} m \log k)$ time for all positions $j$ of $T$, the index of the

**Table 2** The $D$-table for strings $T_1$ = abaab and $T_2$ = babab under the Levenshtein distance. The 2-wave 4, 3, 4, 3, 2 is shown in bold

| | | | b | a | b | a | b |
|---|---|---|---|---|---|---|---|
| | $(i,j)$ | $-1$ | 0 | 1 | 2 | 3 | 4 |
| | $-1$ | 0 | 1 | 2 | 3 | 4 | 5 |
| a | 0 | 1 | 1 | 1 | 2 | 3 | 4 |
| b | 1 | 2 | 1 | 2 | 1 | 2 | 3 |
| a | 2 | 3 | 2 | 1 | 2 | 1 | **2** |
| a | 3 | 4 | 3 | 2 | **2** | 2 | **2** |
| b | 4 | 5 | 4 | 3 | **2** | 3 | **2** |

$k$-th mismatch of $P$ with the suffix $T[j, n-1]$. Applied for $P = T$, it gives the following result.

**Lemma 2** ([22, 29]) *The* PREF$_k$ *table of a string of length n over an integer alphabet can be computed in* $\mathcal{O}(n + \min\{\, nk,\, nk^{2/3}\log^{1/3} n \log k \,\})$ *time.*

We say that strings $U$ and $V$ have a *k-mismatch prefix-suffix of length p* if $U$ has a prefix $U'$ of length $p$ and $V$ has a suffix $V'$ of length $p$ such that $Ham(U', V') \le k$.

## 2.2 Algorithmic Toolbox for Edit Distance

For $x, y \in \Sigma$, let $c(x, y)$, $c(\varepsilon, x)$ and $c(x, \varepsilon)$ be the costs of substituting letter $x$ by letter $y$ (equal to 0 if $x = y$), inserting letter $x$ and deleting letter $x$, respectively. They are usually specified by a penalty matrix $c$; it implies a metric if certain conditions are satisfied (identity of indiscernibles, symmetry, triangle inequality).

In the classic dynamic programming solution to the edit distance problem [35] for strings $T_1$ and $T_2$, a so-called $D$-table is computed such that $D[i, j]$ is the edit distance between prefixes $T_1[0, i]$ and $T_2[0,j]$; see Table 2. Initially $D[-1,-1] = 0$, $D[i,-1] = D[i-1,-1] + c(T_1[i], \varepsilon)$ for $i \ge 0$ and $D[-1,j] = D[-1,j-1] + c(\varepsilon, T_2[j])$ for $j \ge 0$. For $i,j \ge 0$, $D[i,j]$ can be computed as follows:

$$\begin{aligned} D[i,j] = \min(&D[i-1,j-1] + c(T_1[i], T_2[j]),\\ &D[i,j-1] + c(\varepsilon, T_2[j]),\ D[i-1,j] + c(T_1[i], \varepsilon)). \end{aligned}$$

Given a threshold $h$ on the Levenshtein distance, Landau et al. [28] show how to compute the Levenshtein distance between $T_1$ and $bT_2$, for any $b \in \Sigma$, in $\mathcal{O}(h)$ time using previously computed solution for $T_1$ and $T_2$ (another solution was given later by Kim and Park [24]). They define an *h-wave* that contains indices of the last value $h$ in diagonals of the $D$-table. Let $L^h(d) = \max\{i : D[i, i+d] = h\}$. Formally an $h$-wave is:

$$L^h = [L^h(-h), L^h(-h+1), \ldots, L^h(h-1), L^h(h)];$$

see also Table 2. Landau et al. [28] show how to update the $h$-wave when string $T_2$ is prepended by a single letter in $\mathcal{O}(h)$ time. This method was introduced to approximate periodicity in [33].

## 3 Computing *k*-Coverage Under Hamming Distance

Let $T$ be a string of length $n$ and assume that its $\mathsf{PREF}_k$ table is given. We show a linear-time algorithm for computing the $k$-coverage of every prefix of $T$ under the Hamming distance and then apply it to computing the $k$-coverage of all factors.

In the algorithm we consider all prefix lengths $\ell = 1, \dots, n$. At each step of the algorithm, a linked list $\mathcal{L}$ is stored that contains all positions $i$ such that $\mathsf{PREF}_k[i] \geq \ell$ and a sentinel value $n$, in an increasing order. The list is stored together with a table $A(\mathcal{L})[0..n-1]$ such that $A(\mathcal{L})[i]$ is a link to the occurrence of $i$ in $\mathcal{L}$ or **nil** if $i \notin \mathcal{L}$. It can be used to access and remove a given element of $\mathcal{L}$ in $\mathcal{O}(1)$ time. Before the start of the algorithm, $\mathcal{L}$ contains all numbers $0, \dots, n$.

If $i \in \mathcal{L}$ and $j$ is the successor of $i$ in $\mathcal{L}$, then the approximate occurrence of $T[0, \ell - 1]$ at position $i$ accounts for $\min(\ell, j-i)$ positions that are covered in $T$. A pair of adjacent elements $i < j$ in $\mathcal{L}$ is called *overlapping* if $j - i \leq \ell$ and *non-overlapping* otherwise. Hence, each non-overlapping adjacent pair adds the same amount to the number of covered positions.

**Observation 1** *$T[0, \ell - 1]$ is a k-approximate cover of T if and only if $\mathcal{L}$ contains no non-overlapping pairs when the length $\ell$ is considered.*

All pairs of adjacent elements of $\mathcal{L}$ can be conceptually partitioned in two sets, $\mathcal{S}_o$ and $\mathcal{S}_{no}$, of overlapping and non-overlapping pairs, respectively. The sets do not need to be stored explicitly; it suffices to store the following data related to their elements. The non-overlapping pairs $(i, j)$ are stored in buckets that correspond to $j - i$, in a table $B(\mathcal{S}_{no})$ indexed from 1 to $n$. Additionally a table $A(\mathcal{S}_{no})$ indexed 0 through $n - 1$ is stored such that $A(\mathcal{S}_{no})[i]$ points to the location of $(i, j)$ in its bucket, provided that such a pair exists for some $j$, or **nil** otherwise. Finally, the number $num(\mathcal{S}_{no})$ of non-overlapping adjacent pairs is retained. For overlapping adjacent pairs $(i, j)$ only the sum of values $j - i$, $sum(\mathcal{S}_o)$, is stored. Then

$$\mathsf{Covered}_k^{Ham}(T[0, \ell - 1], T) = sum(\mathcal{S}_o) + num(\mathcal{S}_{no}) \cdot \ell. \tag{1}$$

Now we need to describe how the data structures are updated when $\ell$ is incremented; see Fig. 4.

In the algorithm we store a table $Q[0..n]$ of buckets containing pairs $(\mathsf{PREF}_k[i], i)$ grouped by the first component. When $\ell$ changes to $\ell + 1$, the second components of all pairs from $Q[\ell]$ are removed, one by one, from the list $\mathcal{L}$ (using the table $A(\mathcal{L})$).

Let us describe what happens when element $q$ is removed from $\mathcal{L}$. Let $q_1$ and $q_2$ be its predecessor and successor in $\mathcal{L}$. (They exist because 0 and $n$ are never removed from $\mathcal{L}$.) Then each of the pairs $(q_1, q)$ and $(q, q_2)$ is removed from the respective set

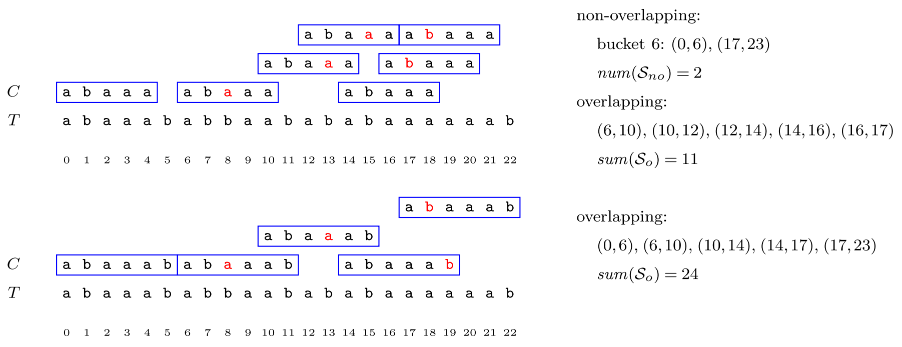

**Fig. 4** Transition from $\ell = 5$ (top) to $\ell = 6$ (bottom) in computing the prefix 1-coverage for an example string. The occurrences at positions 12, 16 are not preserved because their $\mathsf{PREF}_1$ values are equal to 5 (cf. Table 1). Removal of these occurrences changes adjacent pairs (10, 12), (12, 14), (14, 16), (16, 17) to (10, 14), (14, 17), all of which are overlapping. Moreover, the adjacent pairs (0, 6) and (17, 23) become overlapping. For $\ell = 5$ Eq. (1) gives 1-coverage 22, and for $\ell = 6$ we obtain a 1-approximate cover

$\mathcal{S}_o$ or $\mathcal{S}_{no}$, depending on the difference of elements. Removal of a pair $(i, j)$ from $\mathcal{S}_o$ simply consists in decreasing $sum(\mathcal{S}_o)$ by $j - i$, whereas to remove $(i, j)$ from $\mathcal{S}_{no}$ one needs to remove it from the right bucket (using the table $A(\mathcal{S}_{no})$) and decrement $num(\mathcal{S}_o)$. In the end, the pair $(q_1, q_2)$ is inserted to $\mathcal{S}_o$ or to $\mathcal{S}_{no}$ depending on $q_2 - q_1$. Insertion to $\mathcal{S}_o$ and to $\mathcal{S}_{no}$ is symmetric to deletion.

When $\ell$ is incremented, non-overlapping pairs $(i, j)$ with $j - i = \ell + 1$ become overlapping. Thus, all pairs from the bucket $B(\mathcal{S}_{no})[\ell + 1]$ are removed from $\mathcal{S}_{no}$ and inserted to $\mathcal{S}_o$.

This concludes the description of operations on the data structures. Correctness of the resulting algorithm follows from Equation (1). We analyze its complexity in the following theorem.

**Theorem 1** *Let $T$ be a string of length $n$. Assuming that the $\mathsf{PREF}_k$ table for string $T$ is given, the $k$-coverage of every prefix of $T$ under the Hamming distance can be computed in $\mathcal{O}(n)$ time.*

***Proof*** There are up to $n$ removals from $\mathcal{L}$. Initially $\mathcal{L}$ contains $n$ adjacent pairs. Every removal from $\mathcal{L}$ introduces one new adjacent pair, so the total number of adjacent pairs that are considered in the algorithm is $2n - 1$. Each adjacent pair is inserted to $\mathcal{S}_o$ or to $\mathcal{S}_{no}$, then it may be moved from $\mathcal{S}_{no}$ to $\mathcal{S}_o$, and finally it is removed from its set. In total, $\mathcal{O}(n)$ insertions and deletions are performed on the two sets, in $\mathcal{O}(1)$ time each. This yields the desired time complexity of the algorithm. $\square$

Let us note that in order to compute the $k$-coverage of all factors of $T$ that start at a given position $i$, it suffices to use a table $[\mathsf{lcp}_k(i, 0), \ldots, \mathsf{lcp}_k(i, n - 1)]$ instead of $\mathsf{PREF}_k$. Together with Lemma 1 this gives the following result.

**Corollary 1** *Let $T$ be a string of length $n$. The $k$ -coverage of every factor of $T$ under the Hamming distance can be computed in $\mathcal{O}(n^2)$ time.*

## 4 Computing *k*-Coverage Under Edit Distance

Let us state an abstract problem that, to some extent, is a generalization of the $k$-mismatch lcp-queries to the edit distance.

---

LONGEST APPROXIMATE PREFIX PROBLEM

**Input:** A string $T$ of length $n$, a metric $d$ and a number $k$

**Output:** A table $P^d_k$ such that $P^d_k[a, b, a']$ is the maximum $b' \geq a' - 1$ such that $d(T[a, b], T[a', b']) \leq k$ or $-1$ if no such $b'$ exists.

---

***Example 1*** Let us consider the following string of length 12:

| $i$ | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $T[i]$ | a | b | a | a | b | b | b | a | b | b | a | a |

Let us assume that the cost of an insertion and a deletion is 1 and the cost of a substitution is 2. We have $P^{ed}_3[5, 10, 1] = 7$ because $ed(T[5, 10], T[1, 7]) \leq 3$ and $ed(T[5, 10], T[1, b']) > 3$ for $b' > 7$, as shown in the following table:

| $b'$ | 1 | 2 | 3 | 4 | 5 | 6 | **7** | 8 |
|---|---|---|---|---|---|---|---|---|
| $T[1, b']$ | b | ba | baa | baab | baabb | baabbb | baabbba | baabbbab |
| $T[5, 10]$ | bbabba | bbabba | bbabba | bbabba | bbabba | bbabba | bbabba | bbabba |
| $ed(T[5, 10], T[1, b'])$ | 4 | 3 | 4 | 3 | 4 | **3** | 4 | |

| $b'$ | 9 | 10 | 11 |
|---|---|---|---|
| $T[1, b']$ | baabbbabb | baabbbabba | baabbbabbaa |
| $T[5, 10]$ | bbabba | bbabba | bbabba |
| $ed(T[5, 10], T[1, b'])$ | 5 | 4 | 5 |

Having the table $P^d_k$, one can easily compute the $k$-coverage of a factor $T[a, b]$ under metric $d$ as:

$$\mathsf{Covered}^d_k(T[a, b], T) = \left| \bigcup_{a'=0}^{n-1} [a', P^d_k[a, b, a']] \right|, \tag{2}$$

where an interval of the form $[a', b']$ for $b' < a'$ is considered to be empty. The size of the union of $n$ intervals can be computed in $\mathcal{O}(n)$ time, which gives $\mathcal{O}(n^3)$ time over all factors.

***Remark 1*** The set of positions that are covered by occurrences of $T[a, b]$ in $T$ can be computed within the same time complexity as the union of the $n$ intervals.

In Sects. 4.1 and 4.2 we show how to compute the tables $P_k^{Lev}$ and $P_k^{ed}$ for a given threshold $k$ in $\mathcal{O}(n^3)$ and $\mathcal{O}(n^3\sqrt{n\log n})$ time, respectively. Then in Sect. 4.3 we apply the techniques of Sect. 4.2 to obtain an $\mathcal{O}(n^3\sqrt{n\log n})$-time algorithm for computing restricted approximate covers and seeds under the edit distance.

### 4.1 Longest Approximate Prefix Under Levenshtein Distance

Let $H_{i,j}$ be the $h$-wave for strings $T[i, n-1]$ and $T[j, n-1]$ and $h = k$. Then we can compute $P_k^{Lev}$ with Algorithm 1. The algorithm basically takes the rightmost diagonal of $D$-table in which the value in row $b - a + 1$ is less than or equal to $k$.

**Algorithm 1:** Computing $P_k^{Lev}$ table.

```
for $a' := n-1$ down to $0$ do
    Compute $H_{n-1,a'}$;
    for $a := n-1$ down to $0$ do
        if $a < n$ then
            Compute $H_{a,a'}$ from $H_{a+1,a'}$;
        $d := k$;
        for $b := a$ to $n-1$ do
            $i := b - a + 1$;
            while $d \geq -k$ and $H_{a,a'}(d) < i$ do
                $d := d - 1$;
            if $d < -k$ then $P_k^{Lev}[a, b, a'] := -1$;
            else $P_k^{Lev}[a, b, a'] := a' + i + d$;
```

The while-loop can run up to $2k$ times for given $a$ and $a'$. Computing $H_{n-1,a'}$ takes $\mathcal{O}(k^2)$ time and updating $H_{a,a'}$ takes $\mathcal{O}(k)$ time. It makes the algorithm run in $\mathcal{O}(n^3)$ time. Together with Eq. (2) this yields the following result.

**Proposition 1** *Let $T$ be a string of length $n$. The $k$ -coverage of every factor of $T$ under the Levenshtein distance can be computed in $\mathcal{O}(n^3)$ time.*

A similar method could be used in case of constant edit operation costs, by applying the work of [18]. In the following section we develop a solution for arbitrary costs.

### 4.2 Longest Approximate Prefix under Edit Distance

For indices $a, a' \in [0, n]$ we define a table $D_{a,a'}$ such that $D_{a,a'}[b, b']$ is the edit distance between $T[a, b]$ and $T[a', b']$, for $b \in [a-1, n-1]$ and $b' \in [a'-1, n-1]$. For other indices we set $D_{a,a'}[b, b'] = \infty$. The $D_{a,a'}$ table corresponds to the $D$-table for $T[a, n-1]$ and $T[a', n-1]$ and so it can be computed in $\mathcal{O}(n^2)$ time.

We say that pair $(d, b)$ (Pareto-)dominates pair $(d', b')$ if $(d, b) \neq (d', b')$, $d \leq d'$ and $b \geq b'$. Let us introduce a data structure $L_{a,a'}[b]$ being a list of all pairs $(D_{a,a'}[b, b'], b')$ that are maximal in this sense (i.e., are not dominated by other pairs), sorted by increasing first component and stored in a table. Using a folklore stack-based algorithm (Algorithm 2), this data structure can be computed from $D_{a,a'}[b, a'-1], \dots, D_{a,a'}[b, n-1]$ in linear time.

***Example 2*** For the string from Example 1, we have $L_{5,1}[10] = [\,(3,7),\ (4,10),\ (5,11)\,]$.

**Algorithm 2:** Computing $L_{a,a'}[b]$ from $D_{a,a'}[b,\cdot]$.

```
Q := empty stack;
for $b' := a' - 1$ to $n - 1$ do
    $d := D_{a,a'}[b, b']$;
    while $Q$ not empty do
        $(d', x) := top(Q)$;
        if $d' \geq d$ then $pop(Q)$;
        else break;
    $push(Q, (d, b'))$;
$L_{a,a'}[b] := Q$;
```

Every multiple of $M = \lfloor \sqrt{n/\log n} \rfloor$ will be called a *special point*. In our algorithm we first compute the following data structures:

(a) all $L_{a,a'}[b]$ lists where $a$ or $a'$ is a special point, for $a, a' \in [0, n-1]$ and $b \in [a-1, n-1]$ (if $a \geq n$ or $a' \geq n$, the list is empty); and
(b) all cells $D_{a,a'}[b, b']$ of all $D_{a,a'}$ tables for $a, a' \in [0, n]$ and $-1 \leq b - a, b' - a' < M - 1$.

Computing part (a) takes $\mathcal{O}(n^4/M) = \mathcal{O}(n^3\sqrt{n \log n})$ time, whereas part (b) can be computed in $\mathcal{O}(n^4/M^2) = \mathcal{O}(n^3 \log n)$ time. The intuition behind this data structure is shown in the following lemma.

**Lemma 3** *Assume that $b - a \geq M - 1$ or $b' - a' \geq M - 1$. Then there exists a pair of positions $c, c'$ such that the following conditions hold:*

- *$a \leq c \leq b + 1$ and $a' \leq c' \leq b' + 1$, and*
- *$c - a, c' - a' < M$, and*
- *$ed(T[a,b], T[a',b']) = ed(T[a,c-1], T[a',c'-1]) + ed(T[c,b], T[c',b'])$, and*
- *at least one of $c, c'$ is a special point.*

*Moreover, if $c$ ($c'$) is a special point, then $c \leq b$ ($c' \leq b'$, respectively).*

***Proof*** By the assumption, at least one of the intervals $[a, b]$ and $[a', b']$ contains a special point. Let $p \in [a, b]$ and $p' \in [a', b']$ be the smallest among them; we have $p - a, p' - a' < M$ provided that $p$ or $p'$ exists, respectively (otherwise $p$ or $p'$ is set to $\infty$). Let us consider the table $D_{a,a'}$ and how its cell $D_{a,a'}[b, b']$ is computed. We can trace the path of parents in the dynamic programming from $D_{a,a'}[b, b']$ to the origin ($D_{a,a'}[a-1, a'-1]$). Let us traverse this path in the reverse direction until the first dimension of the table reaches $p$ or the second dimension reaches $p'$. Say that just before this step we are at $D_{a,a'}[q, q']$. If $q + 1 = p$ and $q' < p'$, then we set $c = q + 1$ and $c' = q' + 1$. Indeed $c = p$ is a special point,

$$ed(T[a,b], T[a',b']) = ed(T[a,c-1], T[a',c'-1]) + ed(T[c,b], T[c',b'])$$

and $c - a, c' - a' < M$. Moreover, $q' \in [a' - 1, b']$, so $c' \in [a', b' + 1]$. The opposite case (that $q' + 1 = p'$) is symmetric. □

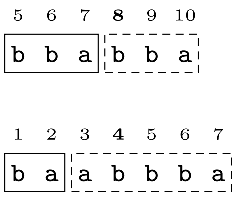

**Fig. 5** Illustration of Example 3

If $b-a<M-1$ and $P^{ed}_k[a,b,a']-a'<M-1$, then $P^{ed}_k[a,b,a']$ can be computed using one of the $M\times M$ prefix fragments of the $D_{a,a'}$ tables. Otherwise, according to the statement of the lemma, one of the $L_{c,c'}[b]$ lists can be used, where $c-a, c'-a'<M$, as shown in Algorithm 3. The algorithm uses a predecessor operation $Pred(x,L)$ which for a number $x$ and a list $L=L_{c,c'}[b]$ returns the maximal pair whose first component does not exceed $x$, or $(\infty,\infty)$ if no such pair exists. This operation can be implemented in $\mathcal{O}(\log n)$ time via binary search.

***Example 3*** Let us consider computing $P^{ed}_3[5,10,1]=7$ for the string $T=$ abaabbbabbaa from Example 1 (insertion and deletion with cost 1, substitution with cost 2) and let us choose $M=4$, so positions 0, 4, 8 are special. In this case $b-a, b'-a'\geq M-1$ and we have

$$ed(T[5,10],T[1,7])=1+2=ed(T[5,7],T[1,2])+ed(T[8,10],T[3,7]),$$

for $c=8, c'=3$ where the position 8 is special, as shown in Fig. 5 (cf. Lemma 3).

One can check that for this text $L_{8,3}[10]=[(2,7),(3,8),(4,9),(5,10),(6,11)]$ holds. In this case $P^{ed}_3[5,10,1]$ is determined by the index in the first element of the list since $D_{5,1}[7,2]+2=3$. Let us emphasize that $D_{5,1}[7,2]$ is stored in the algorithm because $7-5, 2-1<M-1$ and $L_{8,3}[10]$ is stored as 8 is special.

**Algorithm 3:** Computing $P^{ed}_k[a,b,a']$.

```
res := −1;
if $b-a<M-1$ then
    for $b' := a'-1$ to $a'+M-2$ do
        if $D_{a,a'}[b,b'] \leq k$ then
            $res := b'$;
$s := a+((-a) \bmod M)$; $s' := a'+((-a') \bmod M)$;        // closest special points
foreach $(c,c')$ in $(\{s\}\times[a',a'+M-1])\cup([a,a+M-1]\times\{s'\})$ do
    $(d',b') := Pred(k-D_{a,a'}[c-1,c'-1], L_{c,c'}[b])$;
    if $d' \neq \infty$ then
        $res := \max(res,b')$;
$P^{ed}_k[a,b,a'] := res$;
```

**Theorem 2** *Let $T$ be a string of length $n$. The $k$-coverage of every factor of $T$ under the edit distance can be computed in $\mathcal{O}(n^3\sqrt{n\log n})$ time.*

***Proof*** We want to show that Algorithm 3 correctly computes $P^{ed}_k[a,b,a']$. Let us first check that the result $b'=res$ of Algorithm 3 satisfies $D_{a,a'}[b,b']\leq k$. It is clear if the algorithm computes $b'$ in line 5. Otherwise, it is computed in line 10. This means that $L_{c,c'}[b]$ contains a pair $(D_{c,c'}[b,b'],b')$ such that

$$k \ \geq \ D_{c,c'}[b, b'] + D_{a,a'}[c-1, c'-1] \ \geq \ D_{a,a'}[b, b'].$$

Now we show that the returned value *res* is at least $x = P_k^{ed}[a, b, a']$. If $b - a < M - 1$ and $x - a' < M - 1$, then the condition in line 4 holds for $b' = x$, so indeed $res \geq x$. Otherwise, the condition of Lemma 3 is satisfied. The lemma implies two positions $c, c'$ such that at least one of them is special and that satisfy additional constraints.

If $c$ is special, then the constraints $a \leq c$ and $c - a < M$ imply that $c = s$, as defined in line 6. Additionally, $a' \leq c' \leq a' + M - 1$, so $(c, c')$ will be considered in the loop from line 7. By the lemma and the definition of $x$, we have

$$D_{c,c'}[b, x] \ = \ D_{a,a'}[b, x] - D_{a,a'}[c-1, c'-1] \ \leq \ k - D_{a,a'}[c-1, c'-1]. \quad (3)$$

The list $L_{c,c'}[b]$ either contains the pair $(D_{c,c'}[b, x], x)$, or a pair $(D_{c,c'}[b, x'], x')$ such that $D_{c,c'}[b, x'] \leq D_{c,c'}[b, x]$ and $x' > x$. In the latter case by (3) we would have

$$\begin{aligned} k \ &\geq \ D_{a,a'}[c-1, c'-1] + D_{c,c'}[b, x] \\ &\geq \ D_{a,a'}[c-1, c'-1] + D_{c,c'}[b, x'] \ \geq \ D_{a,a'}[b, x'] \end{aligned}$$

and $x' > x$. In both cases the predecessor computed in line 8 returns a value *res* such that $res \geq x$ and $res \neq \infty$. The case that $c'$ is special admits an analogous argument.

Combining Algorithm 3 with Equation (2), we obtain correctness of the computation.

As for complexity, Algorithm 3 computes $P_k^{ed}[a, b, a']$ in $\mathcal{O}(M \log n) = \mathcal{O}(\sqrt{n \log n})$ time and the pre-computations take $\mathcal{O}(n^3 \sqrt{n \log n})$ total time. □

## 4.3 Restricted Approximate Covers and Seeds under Edit Distance

The techniques that were developed in Sect. 4.2 can be used to improve upon the $\mathcal{O}(n^4)$ time complexity of the algorithms for computing the restricted approximate covers and seeds under the edit distance [8, 34]. We describe our solution only for restricted approximate covers; the solution for restricted approximate seeds follows by considering the text $\diamond^{|T|} T \diamond^{|T|}$.

Let us first note that the techniques from the previous subsection can be used as a black box to solve the problem in scope in $\mathcal{O}(n^3 \sqrt{n \log n} \log(nw))$ time, where $w$ is the maximum cost of an edit operation. Indeed, for every factor $T[a, b]$ we use binary search for finding the smallest $k$ for which $T[a, b]$ is a $k$-approximate cover of $T$. A given value $k$ is tested by computing the tables $P_k^{ed}[a, b, a']$ for all $a' = 0, \dots, n-1$ and checking if $\mathsf{Covered}_k^{ed}(T[a, b], T) = n$ using Eq. (2).

Now we proceed to a more efficient solution. Same as in the algorithms from [8, 34] we compute, for every factor $T[a, b]$, a table $Q_{a,b}[0..n]$ such that $Q_{a,b}[i]$ is the minimum edit distance threshold $k$ for which $T[a, b]$ is a $k$-approximate cover of $T[i, n-1]$. In the end, all factors $T[a, b]$ for which $Q_{a,b}[0]$ is minimal need to be reported as restricted approximate covers of $T$. We will show how, given the

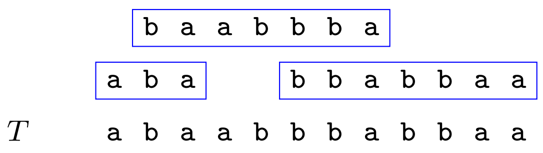

**Fig. 6** Illustration of Example 4

data structures (a) and (b) of the previous section, we can compute this table in $\mathcal{O}(nM \log n)$ time.

***Example 4*** Let us consider the string $T =$ `abaabbbabbaa` from Example 1 (insertion and deletion with cost 1, substitution with cost 2). For the factor $T[5, 10] =$ `bbabba` from this example, we have:

| $i$ | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $T[i]$ | a | b | a | a | b | b | b | a | b | b | a | a |
| $Q_{5,10}[i]$ | 3 | 3 | 3 | 2 | 1 | 1 | 2 | 2 | 2 | 3 | 4 | 5 |

For example, we have $Q_{5,10}[0] = 3$ because $T$ can be covered by factors at edit distance at most 3 from $T[5, 10]$. One of such coverings is depicted in Fig. 6.

A dynamic programming algorithm for computing the $Q_{a,b}$ table, similar to the one in [8], is shown in Algorithm 4. Computing $Q_{a,b}$ takes $\mathcal{O}(n^2)$ time provided that all $D_{a,b}$ arrays, of total size $\mathcal{O}(n^4)$, are available. The algorithm considers all possibilities for the approximate occurrence $T[i, j]$ of $T[a, b]$.

**Algorithm 4:** Computing $Q_{a,b}$ in quadratic time.

```
$Q_{a,b}[n] := 0$;
for $i := n-1$ down to $0$ do
    $Q_{a,b}[i] := \infty$;
    $minQ := \infty$;
    for $j := i$ to $n-1$ do
        $minQ := \min(minQ, Q_{a,b}[j+1])$;          // $minQ = \min Q_{a,b}[i+1..j+1]$
        $Q_{a,b}[i] := \min(Q_{a,b}[i], \max(D_{a,i}[b,j], minQ))$;
```

During the computation of $Q_{a,b}$, we will compute a data structure for on-line range-minimum queries over the table. We can use the following simple data structure with $\mathcal{O}(n \log n)$ total construction time and $\mathcal{O}(1)$-time queries. For every position $i$ and power of two $2^p$, we store as $RM[i,p]$ the minimal value in the table $Q_{a,b}$ on the interval $[i, i+2^p-1]$. When a new value $Q_{a,b}[i]$ is computed, we compute $RM[i,0] = Q_{a,b}[i]$ and $RM[i,p]$ for all $0 < p \le \log_2(n-i)$ using the formula $RM[i,p] = \min(RM[i,p-1], RM[i+2^{p-1}, p-1])$. Then a range-minimum query over an interval $[i, j]$ of $Q_{a,b}$ can be answered by inspecting up to two cells of the $RM$ table for $p$ such that $2^p \le j-i+1 < 2^{p+1}$.

Let us note that the variable $minQ$, which denotes the minimum of a growing segment in the $Q_{a,b}$ table, can only decrease. We would like to make the second argument of max in line 7 non-decreasing for increasing $j$. The values $ed(T[a,b],T[i,j]) = D_{a,i}[b,j]$ may increase or decrease as $j$ grows. However, it is sufficient to consider only those values of $j$ for which $(D_{a,i}[b,j],j)$ is not (Pareto-)dominated (as in Sect. 4.2), i.e., the elements of the list $L_{a,i}[b]$. For these values, $D_{a,i}[b,j]$ is indeed increasing for increasing $j$. The next observation follows from this monotonicity and the monotonicity of $\min Q_{a,b}[i+1..j+1]$.

**Observation 2** *Let $(D_{a,i}[b,j'],j')$ be the first element on the list $L_{a,i}[b]$ such that*

$$\min Q_{a,b}[i+1..j'+1] \le D_{a,i}[b,j'].$$

*If $j'$ does not exist, we simply take the last element of $L_{a,i}[b]$. Further let $(D_{a,i}[b,j''],j'')$ be the predecessor of $(D_{a,i}[b,j'],j')$ in $L_{a,i}[b]$ (if it exists). Then $j \in \{j',j''\}$ minimizes the value of the expression* $\max(\min Q_{a,b}[i+1..j+1], D_{a,i}[b,j])$.

If we had access to the list $L_{a,i}[b]$, we could use binary search to locate the index $j'$ defined in the observation. However, we only store the lists $L_{a,i}[b]$ for $a$ and $i$ such that at least one of them is a special point. We can cope with this issue by separately considering all $j$ such that $j < i + M - 1$ and then performing binary search on every of $\mathcal{O}(M)$ lists $L_{c,c'}[b]$ where $a \le c < a + M$, $i \le c' < i + M$ and at least one of $c$, $c'$ is a special point, just as in Algorithm 3. A pseudocode of the resulting algorithm is given as Algorithm 5.

**Algorithm 5:** Computing $Q_{a,b}$ in $\mathcal{O}(n\sqrt{n \log n})$ time using pre-computed data structures.

```
1  $Q_{a,b}[n] := 0$;
2  for $i := n-1$ down to $0$ do
3      $Q_{a,b}[i] := \infty$;
4      $minQ := \infty$;
5      if $b-a < M-1$ then
6          for $j := i$ to $i+M-2$ do
7              $minQ := \min(minQ, Q_{a,b}[j+1])$;
8              $Q_{a,b}[i] := \min(Q_{a,b}[i], \max(D_{a,i}[b,j], minQ))$;
9      $s := a+((-a) \bmod M)$; $s' := i+((-i) \bmod M)$;
10     foreach $(c,c')$ in $(\{s\} \times [i,i+M-1]) \cup ([a,a+M-1] \times \{s'\})$ do
11         if $L_{c,c'}[b]$ is empty then continue;
           /* Binary search */
12         $(d_{j'},j') :=$ the first pair in $L_{c,c'}[b]$ such that $\min Q_{a,b}[i+1..j'+1] \le D_{a,i}[c-1,c'-1]+d_{j'}$ or the last pair;
13         $(d_{j''},j'') :=$ predecessor of $(d_{j'},j')$ in $L_{c,c'}[b]$ or $(d_{j'},j')$ if there is none;
14         foreach $j$ in $\{j',j''\}$ do
15             $Q_{a,b}[i] := \min(Q_{a,b}[i], \max(D_{a,i}[c-1,c'-1]+d_j, \min Q_{a,b}[i+1..j+1]))$;
```

Let us summarize the complexity of the algorithm. Pre-computation of auxiliary data structures requires $\mathcal{O}(n^3\sqrt{n \log n})$ time. Then for every factor $T[a,b]$ we compute the table $Q_{a,b}$. The data structure for constant-time range-minimum queries over the table costs only additional $\mathcal{O}(n \log n)$ space and computation time. When computing $Q_{a,b}[i]$ using dynamic programming, we may separately consider first $M-1$ indices $j$, and then we perform a binary search in $\mathcal{O}(M)$ lists $L_{c,c'}[b]$. In total, the time to compute $Q_{a,b}[i]$ given $a$, $b$, $i$ is $\mathcal{O}(M \log n) = \mathcal{O}(\sqrt{n \log n})$.

**Theorem 3** *Let $T$ be a string of length $n$. All restricted approximate covers and seeds of $T$ under the edit distance can be computed in $\mathcal{O}(n^3\sqrt{n \log n})$ time.*

The work of [8, 34] on approximate covers and seeds originates from a study of approximate periods [33]. Interestingly, while our algorithm improves upon the algorithms for computing approximate covers and seeds, it does not seem to apply to approximate periods.

## 5 Hardness of Hamming *k*-Approximate Cover and Seed

We consider conditional hardness of the restricted version of the problem and NP-hardness of the general version.

### 5.1 Conditional Hardness of the Restricted Version

We will now show a quadratic conditional lower bound for computing restricted approximate covers. The original problem clearly cannot be solved in subquadratic time because its output has size $\Theta(n^2)$. Hence, we focus on computing restricted approximate covers among factors of a given length. For simplicity we consider a decision version of the problem, in which $k$ is specified.

---

Restricted Approximate Covers of a Given Length

**Input:** String $T$ of length $n$, integers $\ell, k \in [1, n]$, and metric $d$
**Output:** Check, for every length-$\ell$ factor $C$ of $T$, if $C$ is a $k$-approximate cover of $T$ under $d$

---

We will show that existence of a strongly subquadratic-time algorithm for computing restricted approximate covers of a given length for strings over binary alphabet for $k = \Omega(\log n)$ refutes SETH. The hypothesis asserts that for every $\delta > 0$, there exists an integer $q$ such that SAT on $q$-CNF formulas with $m$ clauses and $n$ variables cannot be solved in $m^{O(1)}2^{(1-\delta)n}$ time. Our proof is based on conditional hardness of the following problem.

---

Orthogonal Vectors

**Input:** A set $A$ of $N$ vectors from $\{0,1\}^q$ each
**Output:** Does there exist a pair of vectors $U, V \in A$ that is orthogonal, i.e., $\sum_{i=1}^{q} U[i]V[i] = 0$?

---

**Fact 1** [36, Section 5.1] *Suppose there is $\varepsilon > 0$ such that for all constant $c$, Orthogonal Vectors Problem on a set of $N$ vectors of dimension $q = c \log N$ can be solved in $2^{o(q)} \cdot N^{2-\varepsilon}$ time. Then SETH is false.*

We will make the reduction from a variant of this problem in which there are two sets of $N$ vectors from $\{0,1\}^q$, $A$ and $B$, and we are to check if there is a pair of vectors $U \in A$ and $V \in B$ that are orthogonal. Let us consider two morphisms $\mu$ and $\tau$:

$$\mu(0) = 01011, \quad \mu(1) = 01101, \quad \tau(0) = 10111, \quad \tau(1) = 11101.$$

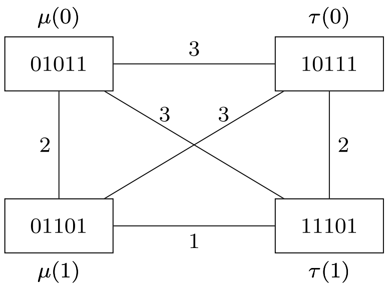

**Fig. 7** Hamming distances between the strings defining morphisms $\mu$ and $\tau$

Let $A' = \{\mu(U) : U \in A\}$ and $B' = \{\tau(V) : V \in B\}$. (Further we treat vectors as strings.) We can make the following easy observation; see Fig. 7.

**Observation 3**

(a) *For every $U, U' \in A'$, $Ham(U, U') \le 2q$. (And for every $V, V' \in B'$, $Ham(V, V') \le 2q$.)*
(b) *Let $U' \in A'$ and $V' \in B'$ such that $U' = \mu(U)$ and $V' = \tau(V)$. If $U$ and $V$ are orthogonal, then $Ham(U', V') = 3q$. Otherwise, $Ham(U', V') \le 3q - 2$.*

Let $G = (0^{5q-1}1)^{3q}$, $A'' = \{UG : U \in A'\}$ and $B'' = \{VG : V \in B'\}$. By $\ell$ we denote the common length of each element of $A'' \cup B''$; we have $\ell = \Theta(q^2)$. Below we use a simple observation that for each $a \in \{0, 1\}$, $\mu(a)$ and $\tau(a)$ contain at least three ones.

**Observation 4** *Let $C = A'' \cup B''$, $U, V, W \in C$, and $Z$ be a length-$\ell$ factor of $VW$. If $Z$ is neither a prefix nor a suffix of $VW$, then $Ham(U, Z) \ge 3q - 1$.*

***Proof*** We consider a few cases depending on the starting position $i$ of an occurrence of $Z$ in $VW$; we have $i \in [1, \ell - 1]$. If $i < 5q$, then $Z$ contains $G$ as a factor being neither its prefix nor its suffix. Then each of the $3q$ 1s in this factor constitutes a mismatch with a 0 in $U$. Similarly, if $i > \ell - 5q$, then each of the $3q$ 1s in the suffix $G$ of $U$ constitutes a mismatch with a 0 in $Z$. Finally, if $5q \le i \le \ell - 5q$, then the length-$5q$ prefix of $U$ contains at least $3q$ 1s, whereas the length-$5q$ prefix of $Z$ is a cyclic shift of $0^{5q-1}1$ and contains exactly one 1. □

The following lemma gives the main part of the reduction.

**Lemma 4** *Consider a set $A = \{W_1, \dots, W_N\}$ of $N$ vectors from $\{0,1\}^q$ with $q \ge 2$, $B = A$, and the sets $A', A'', B', B''$ defined as above with $A'' = \{U_1, \dots, U_N\}$ and $B'' = \{V_1, \dots, V_N\}$. Let*

$$T = U_1 \dots U_N V_1 \dots V_N$$

*and $k = 3q - 2$. Then:*

(a) *If $W_i, W_j \in A$ are orthogonal for $i, j \in [1, N]$, then $U_i$ is not a $k$-approximate cover of $T$.*
(b) *If the set $A$ does not contain two orthogonal vectors, then for each $i \in [1, N]$, $U_i$ is a $k$-approximate cover of $T$.*

***Proof*** By Observation 4, $\mathsf{StartOcc}_k^{Ham}(U_i, T) \subseteq \{0, \ell, 2\ell, \dots, (2N-1)\ell\}$. Hence, $\mathsf{Covered}_k^{Ham}(U_i, T) = |T|$ if and only if this inclusion is an equality.

Let us fix $i \in [1, N]$. If $W_i$ and $W_j$ are orthogonal for some $j \in [1, N]$, then, by Observation 3(b), $(N + j - 1)\ell \notin \mathsf{StartOcc}_k^{Ham}(U_i, T)$, so $\mathsf{Covered}_k^{Ham}(U_i, T) < |T|$.

Now assume that $W_i$ and $W_j$ are not orthogonal for all $j \in [1, N]$. By Observation 3(a), $\{0, \ell, \dots, (N-1)\ell\} \subseteq \mathsf{StartOcc}_k^{Ham}(U_i, T)$, since $2q \le 3q - 2 = k$. By Observation 3(b), we have that $\{N\ell, \dots, (2N-1)\ell\} \subseteq \mathsf{StartOcc}_k^{Ham}(U_i, T)$. In conclusion, $\mathsf{Covered}_k^{Ham}(U_i, T) = |T|$. □

**Theorem 4** *Suppose there is $\varepsilon > 0$ such that* Restricted Approximate Covers of a Given Length *under Hamming distance can be solved in $\mathcal{O}(n^{2-\varepsilon})$ time for strings over binary alphabet. Then SETH is false.*

***Proof*** Let $A$ be an instance of the Orthogonal Vectors problem, with $N$ vectors of dimension $q = \Theta(\log N)$, and text $T$ be defined as in Lemma 4. By the lemma, we can check if $A$ contains a pair of orthogonal vectors by checking which of the length-$\ell$ factors of $T$ are $k$-approximate covers of $T$, for $k = 3q - 2 = \Theta(\log N)$ and $\ell = \Theta(k^2)$. We have $n = |T| = \mathcal{O}(Nq^2)$. If the Restricted Approximate Covers of a Given Length problem can be solved in $\mathcal{O}(n^{2-\varepsilon})$ time, then Orthogonal Vectors can be solved in $\mathcal{O}(N^{2-\varepsilon} \log^{\mathcal{O}(1)} N)$ time, which refutes SETH by Fact 1. □

***Remark 2*** The same conditional lower bound can be proved for Restricted Approximate Covers of a Given Length under edit distance. To show this, it suffices to use the fact that the Levenshtein distance of two binary strings cannot be computed in strongly subquadratic time unless SETH is false [7]. Indeed, in order to check if the Levenshtein distance between binary strings $S_1$ and $S_2$ is at most $k$, one can check if $\#S_1\#$ is a $k$-approximate cover of $T = \#S_1\#S_2\#$, where # is a special symbol such that the cost of any edit operation involving # is $\infty$.

### 5.2 NP-Hardness of the General Version

We make a reduction from the following problem.

Hamming String Consensus

**Input:** Strings $S_1, \dots, S_m$, each of length $\ell$, and an integer $k \le \ell$

**Output:** A string $S$, called consensus string, such that $Ham(S, S_i) \le k$ for all $i = 1, \dots, m$

**Fact 2** [13] Hamming String Consensus *is NP-complete even for the binary alphabet.*

Let strings $S_1, \dots, S_m$ of length $\ell$ over the alphabet $\Sigma = \{0, 1\}$ and integer $k$ be an instance of Hamming String Consensus. We introduce a morphism $\phi$ such that

$$\phi(0) = 0^{2k+4}\,1010\,0^{2k+4}, \quad \phi(1) = 0^{2k+4}\,1011\,0^{2k+4}.$$

We will exploit the following simple property of this morphism.

**Observation 5** *For every string $S$, every length-$(2k+4)$ factor of $\phi(S)$ contains at most three ones.*

Let $\gamma_i = 1^{2k+4}\phi(S_i)$ and let $\psi(U)$ be an operation that reverses this encoding, i.e., $\psi(\gamma_i) = S_i$. Formally, it takes as input a string $U$ and outputs $U[4k+12-1]U[2\cdot(4k+12)-1]\dots U[(\ell-1)(4k+12)-1]$.

**Lemma 5** *Strings $\gamma_i$ and $\gamma_j$, for any $i,j \in \{1, \dots, m\}$, have no $2k$-mismatch prefix-suffix of length $p \in \{2k+4, \dots, |\gamma_i|-1\}$.*

***Proof*** We will show that the prefix $U$ of $\gamma_i$ of length $p$ and the suffix $V$ of $\gamma_j$ of length $p$ have at least $2k+1$ mismatches. Let us note that $U$ starts with $1^{2k+4}$. The proof depends on the value $d = |\gamma_i| - p$; we have $1 \le d \le |\gamma_i| - 2k - 4$. Let us start with the following observation that can be readily verified.

**Observation 6** *For $A, B \in \{1010, 1011\}$, the strings $A0^4$ and $0^4B$ have no 1-mismatch prefix-suffix of length in $\{5, \dots, 8\}$.*

If $1 \le d \le 4$, then $U$ and $V$ have a mismatch at position $2k+4$ since $V$ starts with $1^{2k+4-d}0$. Moreover, they have at least $2\ell$ mismatches by the observation (applied for the prefix-suffix length $d+4$). In total, $Ham(U, V) \ge 2\ell + 1 \ge 2k+1$.

If $4 < d < 2k+4$, then every block 1010 or 1011 in $\gamma_i$ and in $\gamma_j$ is matched against a block of zeroes in the other string, which gives at least $4\ell$ mismatches. Hence, $Ham(U, V) \ge 4\ell \ge 2k+1$.

Finally, if $2k+4 \le d \le |\gamma_i| - 2k - 4$, then $U$ starts with $1^{2k+4}$ and every factor of $V$ of length $2k+4$ has at most three ones (see Observation 5). Hence, $Ham(U, V) \ge 2k+1$. □

We set $T = \gamma_1 \dots \gamma_m$. The following lemma gives the reduction.

**Lemma 6** *If* Hamming String Consensus *for $S_1, \dots, S_m$, $\ell$, $k$ has a positive answer, then the* General $k$ -Approximate Cover *under Hamming distance for $T$, $k$, and $c = |\gamma_i|$ returns a $k$-approximate cover $C$ such that $S = \psi(C)$ is a Hamming consensus string for $S_1, \dots, S_m$.*

***Proof*** By Lemma 5, if $C$ is a $k$-approximate cover of $T$ of length $c$, then every position $a \in \mathsf{StartOcc}_k^H = Ham(C,T)$ satisfies $c \mid a$. Hence, $\mathsf{StartOcc}_k^{Ham}(C,T) = \{0, c, 2c, \dots, (m-1)c\}$.

If Hamming String Consensus for $S_1, \dots, S_m$ has a positive answer $S$, then $1^{2k+4}\phi(S)$ is a $k$-approximate cover of $T$ of length $c$. Moreover, if $T$ has a $k$-approximate cover $C$ of length $c$, then for $S = \psi(C)$ and for each $i = 1, \dots, m$, we have that

$$Ham(C, T[(i-1)c, ic-1]) \geq Ham(S, S_i),$$

so $S$ is a consensus string for $S_1, \dots, S_m$. This completes the proof. □

Lemma 6 and Fact 2 imply that computing $k$-approximate covers is NP-hard. Obviously, it is in NP.

**Theorem 5** General $k$ -Approximate Cover *under the Hamming distance is NP-complete even over a binary alphabet.*

A lemma that is similar to Lemma 6 can be shown for approximate seeds. Let

$$T' = \gamma_1\gamma_1 \cdots \gamma_m 1^{2k+4}\gamma_m 1^{2k+4}.$$

**Lemma 7** *If* Hamming String Consensus *for $S_1, \dots, S_m$, $\ell$, $k$ has a positive answer, then the* General $k$ -Approximate Seed *under Hamming distance for $T'$, $k$, and $c = |\gamma_1| + 2k + 4$ returns a k-approximate seed C such that $S = \psi(C')$ is a Hamming consensus string for $S_1, \dots, S_m$, for some cyclic shift $C'$ of $C$.*

***Proof*** Assume that $C$ is a $k$-approximate seed of $T'$ of length $c$ and let us consider the approximate occurrence of $C$ that covers position $c-1$ in $T'$. Note that it has to be a full occurrence. It follows from the next claim that the position of this occurrence is in $\{0, \dots, 2k+3\} \cup \{|\gamma_1| - 2k - 4, \dots, |\gamma_1| + 2k - 3\}$.

***Claim*** Let $X$ be any length-$c$ factor of $\phi(S_1)1^{2k+4}\phi(S_1)$ and $Y$ be any length-$c$ factor of $(\gamma_m 1^{2k+4})^2$. Then $Ham(X, Y) > 2k$.

***Proof*** Let us note that the string $(\gamma_m 1^{2k+4})^2$ contains a middle block $1^{4k+8}$. If $Y$ contains this whole block, then certainly $Ham(X, Y) \geq 2k+1$, since every factor of $X$ of length $4k+8$ contains at most $2k+7$ ones (see Observation 5). Otherwise,

$$Y = 1^{2k+4+b}\phi(S_m)1^{2k+4-b} \quad \text{or} \quad Y = 1^{2k+4-b}\phi(S_m)1^{2k+4+b}$$

for some $b \in \{0, \dots, 2k+3\}$. In particular, $Y$ has $1^{2k+4}$ as a prefix or as a suffix. By comparing lengths we see that the length-$(2k+4)$ prefix and suffix of $X$ are factors of $\phi(S_1)$. Hence, each of them contains at most three ones (see Observation 5) and $Ham(X, Y) \geq 2k+1$. □

We have established that $C$ has to match, up to at most $k$ mismatches, a string of the form

$$1^b \phi(S_1) 1^{2k+4} 0^{2k+4-b} \quad \text{or} \quad 0^b 1^{2k+4} \phi(S_1) 1^{2k+4-b}$$

for some $b \in \{0, \dots, 2k+4\}$. We consider the second case; a proof for the first case is analogous (using strings $\gamma'_i = \phi(S_i)1^{2k+4}$ instead of $\gamma_i$).

In the second case, $Ham(C, 0^b\gamma_1 1^{2k+4-b}) \le k$. Applying Lemma 5 for $\gamma_1$ and every $\gamma_j$, we get that the starting position $p$ of an occurrence of $C$ in $T'$ that covers the first zero of $\gamma_j$ in the factor $\gamma_1 \dots \gamma_m$ of $T'$ has to satisfy $p \equiv -b \bmod |\gamma_1|$.

If Hamming String Consensus for $S_1, \dots, S_m$ has a positive answer $S$, then the string $1^{2k+4}\phi(S)1^{2k+4}$ is a $k$-approximate cover (hence, $k$-approximate seed) of $T'$ of length $c$. Moreover, if $T'$ has a $k$-approximate seed $C$ of length $c$ such that $Ham(C, 0^b\gamma_1 1^{2k+4-b}) \le k$, then for a cyclic shift $C' = \mathsf{rot}_b(C)$, $S = \psi(C')$ and for each $i = 1, \dots, m$, we have that

$$Ham(C,\ T[i|\gamma_1| - b, (i+1)|\gamma_1| + 2k + 4 - b]) \ge Ham(S, S_i),$$

so $S$ is a consensus string for $S_1, \dots, S_m$. This completes the proof. □

**Theorem 6** General $k$-Approximate Seed *under the Hamming distance is NP-complete even over a binary alphabet.*

## 6 Conclusions

We have presented several polynomial-time algorithms for computing restricted approximate covers and seeds and $k$-coverage under Hamming, Levenshtein and weighted edit distances and shown NP-hardness of non-restricted variants of these problems under the Hamming distance. We have also shown conditional lower bounds for the restricted variants of the problems. However, in many of the problems there is a gap between the (conditional) lower and the current upper bound. An interesting open problem is if restricted approximate covers or seeds under Hamming distance, as defined in [8, 34], can be computed in $\mathcal{O}(n^{3-\varepsilon})$ time, for any $\varepsilon > 0$.

**Acknowledgements** Jakub Radoszewski was supported by the Polish National Science Center, Grant Number 2018/31/D/ST6/03991.

## References

1. Amir, A., Levy, A., Lewenstein, M., Lubin, R., Porat, B.: Can we recover the cover? Algorithmica **81**(7), 2857–2875 (2019). https://doi.org/10.1007/s00453-019-00559-8
2. Amir, A., Levy, A., Lubin, R., Porat, E.: Approximate cover of strings. Theor. Comput. Sci. **793**, 59–69 (2019). https://doi.org/10.1016/j.tcs.2019.05.020
3. Amir, A., Levy, A., Porat, E.: Quasi-periodicity under mismatch errors. In: Navarro, G., Sankoff, D., Zhu, B. (eds.) Annual Symposium on Combinatorial Pattern Matching, CPM 2018, LIPIcs, vol. 105, pp. 4:1–4:15. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik (2018). https://doi.org/10.4230/LIPIcs.CPM.2018.4
4. Apostolico, A., Ehrenfeucht, A.: Efficient detection of quasiperiodicities in strings. Theor. Comput. Sci. **119**(2), 247–265 (1993). https://doi.org/10.1016/0304-3975(93)90159-Q
5. Apostolico, A., Farach, M., Iliopoulos, C.S.: Optimal superprimitivity testing for strings. Inf. Process. Lett. **39**(1), 17–20 (1991). https://doi.org/10.1016/0020-0190(91)90056-N
6. Breslauer, D.: An on-line string superprimitivity test. Inf. Process. Lett. **44**(6), 345–347 (1992). https://doi.org/10.1016/0020-0190(92)90111-8
7. Bringmann, K., Künnemann, M.: Quadratic conditional lower bounds for string problems and dynamic time warping. In: Guruswami, V. (ed.) IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015, pp. 79–97. IEEE Computer Society (2015). https://doi.org/10.1109/FOCS.2015.15
8. Christodoulakis, M., Iliopoulos, C.S., Park, K., Sim, J.S.: Approximate seeds of strings. J. Autom. Lang. Comb. **10**(5/6), 609–626 (2005)
9. Crochemore, M., Rytter, W.: Jewels of Stringology. World Scientific, Singapore (2003). https://doi.org/10.1142/4838
10. Czajka, P., Radoszewski, J.: Experimental evaluation of algorithms for computing quasiperiods. Theor. Comput. Sci. **854**, 17–29 (2021). https://doi.org/10.1016/j.tcs.2020.11.033
11. Flouri, T., Giaquinta, E., Kobert, K., Ukkonen, E.: Longest common substrings with k mismatches. Inf. Process. Lett. **115**(6–8), 643–647 (2015). https://doi.org/10.1016/j.ipl.2015.03.006
12. Flouri, T., Iliopoulos, C.S., Kociumaka, T., Pissis, S.P., Puglisi, S.J., Smyth, W.F., Tyczyński, W.: Enhanced string covering. Theor. Comput. Sci. **506**, 102–114 (2013). https://doi.org/10.1016/j.tcs.2013.08.013
13. Frances, M., Litman, A.: On covering problems of codes. Theory Comput. Syst. **30**(2), 113–119 (1997). https://doi.org/10.1007/s002240000044
14. Guth, O.: Searching regularities in strings using finite automata. Ph.D. thesis, Czech Technical University in Prague (2014)
15. Guth, O.: On approximate enhanced covers under Hamming distance. Discrete Appl. Math. **274**, 67–80 (2020). https://doi.org/10.1016/j.dam.2019.01.015
16. Guth, O., Melichar, B.: Using finite automata approach for searching approximate seeds of strings. In: Huang, X., Ao, S.I., Castillo, O. (eds.) Intelligent Automation and Computer Engineering, pp. 347–360. Springer, Netherlands (2010). https://doi.org/10.1007/978-90-481-3517-2_27
17. Guth, O., Melichar, B., Balík, M.: Searching all approximate covers and their distance using finite automata. In: P. Vojtás (ed.) Proceedings of the Conference on Theory and Practice of Information Technologies, ITAT 2008, CEUR Workshop Proceedings, vol. 414. CEUR-WS.org (2008). http://ceur-ws.org/Vol-414/paper4.pdf
18. Hyyrö, H., Narisawa, K., Inenaga, S.: Dynamic edit distance table under a general weighted cost function. J. Discrete Algorithms **34**, 2–17 (2015). https://doi.org/10.1016/j.jda.2015.05.007
19. Iliopoulos, C.S., Moore, D.W.G., Park, K.: Covering a string. Algorithmica **16**(3), 288–297 (1996). https://doi.org/10.1007/BF01955677
20. Impagliazzo, R., Paturi, R.: On the complexity of k-SAT. J. Comput. Syst. Sci. **62**(2), 367–375 (2001). https://doi.org/10.1006/jcss.2000.1727
21. Impagliazzo, R., Paturi, R., Zane, F.: Which problems have strongly exponential complexity? J. Comput. Syst. Sci. **63**(4), 512–530 (2001). https://doi.org/10.1006/jcss.2001.1774
22. Kaplan, H., Porat, E., Shafrir, N.: Finding the position of the k-mismatch and approximate tandem repeats. In: L. Arge, R. Freivalds (eds.) Algorithm Theory—SWAT 2006, 10th Scandinavian Workshop on Algorithm Theory, Lecture Notes in Computer Science, vol. 4059, pp. 90–101. Springer (2006). https://doi.org/10.1007/11785293_11

23. Kędzierski, A., Radoszewski, J.: k-approximate quasiperiodicity under Hamming and edit distance. In: I.L. Gørtz, O. Weimann (eds.) 31st Annual Symposium on Combinatorial Pattern Matching, CPM 2020, *LIPIcs*, vol. 161, pp. 18:1–18:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2020). https://doi.org/10.4230/LIPIcs.CPM.2020.18
24. Kim, S., Park, K.: A dynamic edit distance table. J. Discrete Algorithms **2**(2), 303–312 (2004). https://doi.org/10.1016/S1570-8667(03)00082-0
25. Kociumaka, T., Kubica, M., Radoszewski, J., Rytter, W., Waleń, T.: A linear-time algorithm for seeds computation. ACM Trans. Alg. **16**(2), Article 27 (2020). https://doi.org/10.1145/3386369
26. Kociumaka, T., Pissis, S.P., Radoszewski, J., Rytter, W., Waleń, T.: Efficient algorithms for shortest partial seeds in words. Theor. Comput. Sci. **710**, 139–147 (2018). https://doi.org/10.1016/j.tcs.2016.11.035
27. Kociumaka, T., Pissis, S.P., Radoszewski, J., Rytter, W., Waleń, T.: Fast algorithm for partial covers in words. Algorithmica **73**(1), 217–233 (2015). https://doi.org/10.1007/s00453-014-9915-3
28. Landau, G.M., Myers, E.W., Schmidt, J.P.: Incremental string comparison. SIAM J. Comput. **27**(2), 557–582 (1998). https://doi.org/10.1137/S0097539794264810
29. Landau, G.M., Vishkin, U.: Efficient string matching with k mismatches. Theor. Comput. Sci. **43**, 239–249 (1986). https://doi.org/10.1016/0304-3975(86)90178-7
30. Li, Y., Smyth, W.F.: Computing the cover array in linear time. Algorithmica **32**(1), 95–106 (2002). https://doi.org/10.1007/s00453-001-0062-2
31. Moore, D.W.G., Smyth, W.F.: An optimal algorithm to compute all the covers of a string. Inf. Process. Lett. **50**(5), 239–246 (1994). https://doi.org/10.1016/0020-0190(94)00045-X
32. Moore, D.W.G., Smyth, W.F.: A correction to "An optimal algorithm to compute all the covers of a string." Inf. Process. Lett. **54**(2), 101–103 (1995). https://doi.org/10.1016/0020-0190(94)00235-Q
33. Sim, J.S., Iliopoulos, C.S., Park, K., Smyth, W.F.: Approximate periods of strings. Theor. Comput. Sci. **262**(1), 557–568 (2001). https://doi.org/10.1016/S0304-3975(00)00365-0
34. Sim, J.S., Park, K., Kim, S.R., Lee, J.S.: Finding approximate covers of strings. J. Korea Inf. Sci. Soc. **29**(1), 16–21 (2002)
35. Wagner, R.A., Fischer, M.J.: The string-to-string correction problem. J. ACM **21**(1), 168–173 (1974)
36. Williams, R.: A new algorithm for optimal 2-constraint satisfaction and its implications. Theor. Comput. Sci. **348**(2–3), 357–365 (2005). https://doi.org/10.1016/j.tcs.2005.09.023