Approximate Cover of Strings

Amihood Amir[1], Avivit Levy[2], Ronit Lubin[3], and Ely Porat[4]

1 **Bar-Ilan University, Ramat Gan, Israel; and**
Johns Hopkins University, Baltimore, MD, USA
amir@cs.biu.ac.il

2 **Shenkar College, Ramat Gan, Israel**
avivitlevy@shenkar.ac.il

3 **Bar-Ilan University, Ramat Gan, Israel**
ronit.moldovan@gmail.com

4 **Bar-Ilan University, Ramat Gan, Israel**
porately@cs.biu.ac.il

Abstract

Regularities in strings arise in various areas of science, including coding and automata theory, formal language theory, combinatorics, molecular biology and many others. A common notion to describe regularity in a string T is a *cover*, which is a string C for which every letter of T lies within some occurrence of C. The alignment of the cover repetitions in the given text is called *a tiling*. In many applications finding exact repetitions is not sufficient, due to the presence of errors. In this paper, we use a new approach for handling errors in coverable phenomena and define the *approximate cover problem (ACP)*, in which we are given a text that is a sequence of some cover repetitions with possible mismatch errors, and we seek a string that covers the text with the minimum number of errors. We first show that the ACP is $\mathcal{NP}$-hard, by studying the *cover-size relaxation of the ACP*, in which the requested size of the approximate cover is also given with the input string. We show this relaxation is already $\mathcal{NP}$-hard. We also study another two relaxations of the ACP, which we call the *partial-tiling relaxation of the ACP* and the *full-tiling relaxation of the ACP*, in which a tiling of the requested cover is also given with the input string. A given full tiling retains all the occurrences of the cover before the errors, while in a partial tiling there can be additional occurrences of the cover that are not marked by the tiling. We show that the partial-tiling relaxation has a polynomial time complexity and give experimental evidence that the full-tiling also has polynomial time complexity. The study of these relaxations, besides shedding another light on the complexity of the ACP, also involves a deep understanding of the properties of covers, yielding some key lemmas and observations that may be helpful for a future study of regularities in the presence of errors.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems, G.2.1 Combinatorics, G.4 Mathematical Software, I.5.2 Design Methodology

Keywords and phrases periodicity, quasi-periodicity, cover, approximate cover

Digital Object Identifier 10.4230/LIPIcs.CPM.2017.26

1 Introduction

Regularities in strings arise in various areas of science, including coding and automata theory, formal language theory, combinatorics, molecular biology and many others. A typical form of regularity is *periodicity*, meaning that a "long" string T can be represented as a concatenation of copies of a "short" string P, possibly ending in a prefix of P. Periodicity has been extensively studied in Computer Science over the years (see [26]).

For many phenomena, it is desirable to broaden the definition of periodicity and study wider classes of repetitive patterns in strings. One common such notion is that of a *cover*, defined as follows.

▶ **Definition 1** (Cover). A length m substring C of a string T of length n, is said to be a *cover* of T, if $n > m$ and every letter of T lies within some occurrence of C.

Note that by the definition of cover, the string C is both a prefix and a suffix of the string T. For example, consider the string $T = abaababaaba$. Clearly, T is "almost" periodic with period aba, however, as it is not completely periodic, the algorithms that exploit repetitions cannot be applied to it. On the other hand, the string $C = aba$ is a cover of T, which allows applying to T cover-based algorithms. In this paper we study coverable phenomena in the presence of errors.

There are related regularity types and several approaches to handle errors in regularities. Quasi-periodicity was introduced by Ehrenfeucht in 1990 (according to [5]). The earliest paper in which it was studied is by Apostolico, Farach and Iliopoulos [7], which defined the *quasi-period* of a string to be the length of its shortest cover and presented an algorithm for computing the quasi-period of a given string in $O(n)$ time and space. The new notion attracted immediately several groups of researchers (e.g. [8], [27, 28], [25], [9]). An overview on the first decade of the research on covers can be found in the surveys [5, 19, 30].

While covers are a significant generalization of the notion of periods as formalizing regularities in strings, they are still restrictive, in the sense that it remains unlikely that an arbitrary string has a cover shorter than the word itself. Due to this reason, different variants of quasi-periodicity have been introduced. These include *seeds* [18], *maximal quasi-periodic substring* [6], the notion of *k-covers* [20], *λ-cover* [31], *enhanced covers* [15], *partial cover* [21]. Other recently explored directions include the inverse problem for cover arrays [13], extensions to strings in which not all letters are uniquely defined, such as *indeterminate strings* [4] or *weighted sequences* [32]. Some of the related problems are $\mathcal{NP}$-hard (see e.g., [4, 10, 21]).

In applications such as molecular biology and computer-assisted music analysis, finding exact repetitions and covers is not always sufficient. A more appropriate notion is that of *approximate repetitions*, where errors are allowed (see, e.g., [12, 14]). This notion was first studied in 1993 by Landau and Schmidt [23, 24] who concentrated on approximate tandem repeats. Note that, the natural definition of an approximate repetition is not clear. One possible definition is that the distance between any two adjacent repeats is small. Another possibility is that all repeats lie at a small distance from a single "original". Such a definition of *approximate seeds* is studied in [10, 17, 16]. Indeed, all these definitions along with other ones were proposed and studied (see [1, 22, 29]). Yet another possibility is that all repeats must be equal, but we allow a fixed total number of mismatches. The possibility presented in [1] is a global one, assuming that an original unknown string is a sequence of repeats without errors, but the process of sequence creation or transmission incurs errors to the sequence of repeats, and, thus, the examined input string is not a sequence of repeats. Therefore, a (smallest) repeat generating a string with the minimum total number of mismatches with the input string is sought. Extension of this definition approach to approximate covers is the topic of this paper.

1.1 Our Results

In this paper we extend the approach of [1] to the notion of covers and define the *approximate cover problem (ACP)*, in which we are given a text that is a sequence of some cover repetitions with possible mismatch errors, and we seek a string that covers the text with the minimum number of errors. The alignment of the cover repetitions in the given text is called *a tiling*. We prove that the ACP is $\mathcal{NP}$-hard by studying a relaxation of this problem, which we call *the cover-size relaxation of the ACP*. In this relaxation the requested size of the approximate cover is also given with the input string. We prove that this relaxation is already $\mathcal{NP}$-hard, thus proving the $\mathcal{NP}$-hardness of ACP.

We also study another two relaxations of the problem, which we call *the partial-tiling relaxation of the ACP* and *the full-tiling relaxation of the ACP*. In this relaxations a tiling of the requested cover is also given, and we seek a string such that when using the given tiling to align it with the given text, the number of mismatches is minimized. The full tiling retains all the occurrences of the cover before the errors, while in the partial tiling there can be additional occurrences of the cover that are not marked by the tiling. We examine these relaxations and show the partial-tiling has polynomial time complexity and give experimental evidence that the full-tiling also has polynomial time complexity. The study of these relaxations, besides shedding another light on the complexity of the ACP, also involves a deep understanding of the properties of covers and seeds, yielding some key lemmas and observations (such as [2]) that may be helpful for a future study of regularities in the presence of errors.

Paper Contributions. The main contributions of this paper are:

- Proving that the ACP is $\mathcal{NP}$-hard.
- Formalizing the partial-tiling relaxation of the ACP and proving it is polynomial time computable.
- Formalizing the full-tiling relaxation of the ACP and suggesting a polynomial time algorithm for its computation, while giving an experimental evidence for the correctness of this algorithm.

The paper is organized as follows. In Section 2, we give formal definitions. In Section 3, we study the cover-size relaxation of the ACP and prove the $\mathcal{NP}$-hardness of the ACP. In Section 4, we study the partial-tiling relaxation of the ACP and show it is polynomial-time computable. In Section 5, we study the full-tiling relaxation of the ACP, suggest a polynomial-time algorithm for this problem and experimentally test its correctness. We conclude with some open problems in Section 6.

2 Preliminaries

In this section we give the needed formal definitions.

▶ **Definition 2** (Tiling). Let T be a string over alphabet Σ such that the string C over alphabet Σ is a cover of T. Then, the sorted list of indices representing the start positions of occurrences of the cover C in the text T is called the *tiling* of C in T.

In this paper we have a text T which may have been introduced to errors and, therefore, is not coverable. However, we would like to refer to a retained tiling of an unknown string C in T although C does not cover T because of mismatch positions. The following definition makes a distinction between a list of indices that may be assumed to be a tiling of the text before mismatch errors occurred and a list of indices that cannot be such a tiling.

▶ **Definition 3** (A Valid Tiling). Let T be an n-length string over alphabet Σ and let L be a sorted list of indices $L \subset \{1, ..., n\}$. Let $m = n + 1 - L_{last}$, where L_{last} is the last index in L. Then, L is called *a valid tiling* of T, if $i_1 = 1$ and for every $i_k, i_{k+1} \in L$, it holds that $i_{k+1} - i_k \leq m$.

▶ **Notation 1.** *Let C be an m length string over alphabet Σ. Denote by $S(C)$ a string of length n, $n > m$, such that C is a cover of $S(C)$.*

Note that $S(C)$ is not uniquely defined even for a fixed $n > m$, since every different valid tiling of the m-length string C generates a different n-length string $S(C)$. A unique version can be obtained if a valid tiling L is also given.

▶ **Notation 2.** *Let T be an n-length string over alphabet Σ and let L be a valid tiling of T. Let $m = n + 1 - L_{last}$, where L_{last} is the last index in the tiling L. For any m-length string C', let $S_L(C')$ be the n-length string obtained using C' as a cover and L as the tiling as follows: $S_L(C')$ begins with a copy of C' and for each index i in L a new copy of C' is concatenated starting from index i of $S_L(C')$ (maybe running over a suffix of the last copy of C').*

▶ **Definition 4.** Let T be a string of length n over alphabet Σ. Let H be the Hamming distance. The *distance of T from being covered* is:

$$dist = \min_{C\in\Sigma^*, |C|<n, S(C)\in\Sigma^n} H(S(C), T).$$

We will also refer to $dist$ as *the number of errors in T*.

▶ **Definition 5.** Let T be an n-long string over alphabet Σ. An m-long string C over Σ, $m \in \mathbb{N}$, $m < n$, is called *an m-length approximate cover of T*, if for every string C' of length m over Σ, $\min_{S(C')\in\Sigma^n} H(S(C'), T) \geq \min_{S(C)\in\Sigma^n} H(S(C), T)$, where H is the hamming distance of the given strings.
We refer to $\min_{S(C)\in\Sigma^n} H(S(C), T)$ as the *number of errors of an m-length approximate cover of T*.

▶ **Definition 6** (Approximate Cover)**.** Let T be a string of length n over alphabet Σ. A string C over alphabet Σ is called an *approximate cover of T* if:

1. C is an m-length approximate cover of T for some $m \in \mathbb{N}$, $m < n$, for which

$$\min_{S(C)\in\Sigma^n} H(S(C), T) = dist.$$

2. for every m'-length approximate cover of T, C', s.t. $\min_{S(C')\in\Sigma^n} H(S(C'), T) = dist$, it holds that: $m' \geq m$.

Primitivity. By definition, an approximate cover C should be *primitive*, i.e., it cannot be covered by a string other than itself (otherwise, T has a cover with a smaller length). Note that a periodic string can be covered by a smaller string (not necessarily the period), and therefore, is not primitive.

▶ **Definition 7.** The *Approximate Cover Problem* (ACP) is the following:
INPUT: String T of length n over alphabet Σ.
OUTPUT: An approximate cover of T, C, and the number of errors in T.

3 $\mathcal{NP}$-Hardness of the ACP

In this section we prove the $\mathcal{NP}$-hardness of the ACP. To this end, we study a variant of the problem where m, the length of a requested approximate cover, is also given together with the input string T, and we are requested to find a string C of length m that is an

m-length approximate cover of T, i.e., C covers T with the minimum number of errors over all strings of length m. We call this problem *the cover-size relaxation of the ACP*. Clearly, if the cover-size relaxation of the ACP is already $\mathcal{NP}$-hard, then so is the ACP.

Our hardness proof uses a reduction from the 3-SAT problem, in which the input is a logical formula φ on N variables in 3-CNF (each clause has exactly three literals), and we need to decide whether φ is satisfiable or not. The $\mathcal{NP}$-hardness of 3-SAT is well-known (see e.g. [11]).

3.1 The Reduction from 3-SAT

Given a 3-CNF formula φ on N variables, $x_1, \ldots, x_N$, with ℓ clauses. Assume without loss of generality that the literals in each clause are sorted by the index of their variables. We need to define a text T of length n over an alphabet Σ and to specify the size m of the requested approximate cover. We will then show that φ is satisfiable if and only if T has an m-approximate cover with at most some specified number of errors to be defined.

We begin by defining the alphabet Σ to include all the variables and their negation together with 4 additional dummy variables: $x_0, x_{-1}, x_{N+1}, x_{N+2}$ and also a special padding character p. Formally,

$$\Sigma = \{x_i, \bar{x_i} | i \in [1..N]\} \cup \{x_{-1}, x_0, x_{N+1}, x_{N+2}, p\}.$$

The definition of the text T has two parts: a header and a body, where the body of T is defined according to the clauses of the given logical formula φ, and the header preceding this body imposes a structure on an m-approximate cover for T.

The definition of the body of T follows directly from the formula φ. For each clause $C_j = L_1^j \vee L_2^j \vee L_3^j$ of φ, $1 \leq j \leq \ell$, we add to the body of T the substring $L_1^j L_2^j L_3^j$, preceded and followed by a padding of $2N + 14$ occurrences of the character p. The role of this padding is to avoid overlaps between occurrences of an approximate cover covering substrings originating from different clauses. The header is composed of $\ell(N+3)$ copies of the following string: $B = p \ldots p x_{N+2} x_{N+1} \bar{x_N} \ldots \bar{x_1} x_0 x_{-1} p \ldots p x_{N+2} x_{N+1} x_N \ldots x_1 x_0 x_{-1} p \ldots p$, where each padding contains $N + 7$ occurrences of p.

We define the size of the requested approximate cover m to be $3N + 18$. Note that the size of T and m as well as their construction are polynomial in N and ℓ. Lemma 8 assures the correctness of the reduction.

▶ **Lemma 8.** *φ is satisfiable if and only if T has an m-approximate cover with at most $\ell(N + 3)(N + 1)$ errors.*

We have, therefore, proven Theorem 9.

▶ **Theorem 9.** *ACP is $\mathcal{NP}$-hard.*

4 The Partial-Tiling Relaxation of the ACP

In this section we study another relaxation of the approximate cover problem: the partial-tiling relaxation, in which we are given a retained tiling of the cover before the errors has occurred together with the input string itself. In order to formally define the relaxation we need Definitions 10 and 11.

▶ **Definition 10.** Let T be an n-length string over alphabet Σ and let L be a valid tiling of T. Let $m = n + 1 - L_{last}$, where L_{last} is the last index in the tiling L. Then, an

L-approximate cover of T is a primitive string C such that for every string C' of length m over Σ, $H(S_L(C'),T) \geq H(S_L(C),T)$, where H is the hamming distance of the given strings. $\min_{C\in\Sigma^m} H(S_L(C),T)$ is the number of errors of an L approximate cover of T.

▶ **Definition 11.** Let T be an n-length string over alphabet Σ. Let L be a valid tiling of T and let L' be a valid tiling of T such that $L \subseteq L'$. Let $m' = n+1-L'_{last}$, where L'_{last} is the last index in the tiling L'. Then, a *partial L-approximate cover of* T is a primitive string C of length m' such that for every string C' of length m' over Σ, $H(S_{L'}(C'),T) \geq H(S_{L'}(C),T)$, where H is the hamming distance of the given strings.
$\min_{C\in\Sigma^{m'}} H(S_{L'}(C),T)$ is the number of errors of a partial L-approximate cover of T.

▶ **Definition 12** (The Partial-Tiling Relaxation of the ACP).
INPUT: String T of length n over alphabet Σ, and a valid tiling L of T.
OUTPUT: A partial L-approximate cover C of T.

We describe an algorithm for the partial-tiling relaxation of the approximate cover problem in two parts. We first describe the mandatory part of the algorithm, which we call the Histogram Greedy Algorithm. This algorithm does the main work in finding an approximate cover subject to the tiling L. It returns a candidate for the final L approximate cover to be output. This candidate is legal if it is primitive and illegal, otherwise. We then describe the second part, which we call the Partial-Tiling Primitivity Coercion. In this part, the legality of the candidate is checked, and if needed, the candidate is corrected in order to coerce the primitivity requirement.

4.1 The Histogram Greedy Algorithm

This part of the algorithm performs the following steps given the text T and the valid tiling L:

1. Find m, the length of an approximate cover subject to the tiling L, by computing the difference between $n+1$, and the last index in the tiling L, L_{last}, which indicates the last occurrence of the cover in T.
2. Compute the m-length mask M of an approximate cover, by initializing M to zeroes, setting $M[1]=1$, then reading the tiling L from beginning to end and for each $i_k, i_{k+1} \in L$ setting $M[i_{k+1}-i_k]=1$.
3. Compute the m-long string V_C of variables from an auxiliary alphabet

$$\Sigma_V = \{v_1, v_2, \ldots, v_m\}.$$

   First, we initialize the m-long string V_C to $v_1v_2\ldots v_m$. Then, we read the mask M from end to beginning, and for every j such that $M[j]=1$, we update the string V_C by equalizing the substrings $V_C[1..m-j+1]$ and $V_C[j..m]$. In the equalization process, when we obtain an equation $v_k = v_\ell$ for $k<\ell$, we replace both letters by v_k. The resulting string V_C represents C in the following sense: for any pair of indices $1 \leq i < j \leq m$, if $V_C[i] = V_C[j]$ then $C[i] = C[j]$. However, it can be that $V_C[i] \neq V_C[j]$, while $C[i] = C[j]$. In other words, V_C carries the information on equalities imposed by the mask M between indices of C.
4. Compute the n-long string V_T of variables from the auxiliary alphabet Σ_V, which is a string covered by V_C according to the tiling L of T. V_C is computed using the tiling L and V_C as follows: it begins with a copy of V_C and for each index i in L a new copy of V_C is concatenated starting from index i of V_T (maybe running over a suffix of the last copy of V_C).

5. Compute the histogram $Hist_{V_C,\Sigma}$ using the alignment of T with V_T and counting for each variable $V \in V_C$ and each $\sigma \in \Sigma$, the number of indices i in T, V_T for which $V_T[i] = V$ and $T[i] = \sigma$.
6. Compute an L approximate cover candidate C greedily according to the histogram $Hist_{V_C,\Sigma}$, as follows: for every index $1 \le i \le m$, set $C[i] = \sigma_0$, where $Hist_{V_C,\Sigma}[V_C[i], \sigma_0] = \max_{\sigma\in\Sigma} Hist_{V_C,\Sigma}[V_C[i], \sigma]$, i.e., for each index in C we choose the alphabet symbol that minimizes the number of mismatch errors between $S_L(C)$ and T in the relevant indices according to the tiling L.

The algorithm outputs the m-length string C from its last step and the histogram table $Hist_{V_C,\Sigma}$.

Lemma 13 describes a property of the output C returned by the Histogram Greedy algorithm, and immediately follows from the greedy criterion used in step 6 of the algorithm. Lemma 14 describes the algorithm time complexity.

▶ **Lemma 13.** *Let C be the output of the Histogram Greedy algorithm. Then,*

$$H(T, S_L(C)) = \min_{C' \in \Sigma^m} H(T, S_L(C')).$$

▶ **Lemma 14.** *The time complexity of the Histogram Greedy algorithm is: $O(|\Sigma| \cdot m + n)$.*

Despite Lemma 13, the output C of the Histogram Greedy algorithm might not be an L approximate cover of T, because it might not be primitive, as the following example shows.

Example: Assume that $V_C = XYZWXY$ and $\Sigma = \{a, b\}$ and that the histogram $Hist_{V_C,\Sigma}$ computed by the algorithm is the following:

$V_C \diagdown \Sigma$	a	b
X	4	1
Y	2	3
Z	2	1
W	0	3

Then, the Histogram Greedy algorithm chooses: $X = a$, $Y = b$, $Z = a$, $W = b$, and outputs $C = ababab$, which cannot be considered a legal cover since it is not primitive, i.e., C itself can be covered by the shorter string ab. However, the partial L-approximate cover can have a tiling L', such that $L \subseteq L'$, which exactly is the case with ab. Therefore, ab should be returned as the partial L-approximate cover of T. The Partial-Tiling Primitivity Coercion algorithm described in Subsection 4.2 is responsible for checking the legality of the output string received from the Histogram Greedy algorithm and returning a partial L-approximate cover.

Note, that the input tiling L requires an m-length string as an output. Therefore, the (primitive) 2-length approximate cover ab is precluded as an L-approximate cover. Assuming that the input tiling L is the retained tiling of the cover of the original text before the errors occurred, such a case means that, though ab is a string covering T subject to a partial tiling L with the least number of errors, it does not cover T with L as a full tiling. In this sense, L is an evidence that the original cover is of larger length than ab and that more errors actually happened. Section 5 is devoted to finding an L-approximate cover.

4.2 The Partial-Tiling Primitivity Coercion Algorithm

This part of the algorithm gets as input the string C returned by the Histogram Greedy algorithm and performs the following steps:

1. Check the primitivity of C (using the linear-time algorithm of [7]). If C is primitive, return C.
2. Else, return the primitive cover C' of C (found using the linear-time algorithm of [7] in the first step).

The time complexity of the Partial-Tiling Primitivity Coercion algorithm is immediate from the linear-time complexity of the algorithm in [7]. Thus, we get:

▶ **Lemma 15.** *The time complexity of the Partial-Tiling Primitivity Coercion algorithm is $O(m)$.*

Theorem 16 follows.

▶ **Theorem 16.** *Given a text T of length n over alphabet Σ and a valid tiling L. Let L_{last} be the last index in L. Then, the partial-tiling relaxation of the approximate cover problem of T can be solved in $O(|\Sigma| \cdot m + n)$ time, where $m = n + 1 - L_{last}$.*

5 The Full-Tiling Relaxation of the ACP

In this section we study another relaxation of the approximate cover problem: the full-tiling relaxation, in which we are given a retained tiling of the cover before the errors have occurred together with the input string itself. Unlike the situation in the problem of the previous section, this tiling is assumed to be exact. Therefore, the algorithm cannot return as cover a string that in order to cover T must have repetitions that are not marked in the tiling L. The formal definition of the problem is as follows.

▶ **Definition 17** (The Full-Tiling Relaxation of the ACP).
INPUT: String T of length n over alphabet Σ, and a valid tiling L of T.
OUTPUT: An L-approximate cover C of T.

In order to impose the requirement of the definition of an L-approximate cover of T to be a primitive string such that all its repetitions to cover T (with minimum number of errors) are marked in the tiling L, we need a different primitivity coercion algorithm than the one described in the previous section. This algorithm is described in Subsection 5.1. Unfortunately, proving the correctness of this algorithm requires a deep understanding of the properties of coverability in the presence of mismatch errors. Although we are making progress in proving this needed background (see, for example [2]), a lack in the complete understanding of the phenomenon prevents us from proving the correctness formally. Hence, in Subsection 5.2, we resort to experimental evidence of the correctness.

5.1 The Full-Tiling Primitivity Coercion Algorithm

This part of the algorithm gets as input the string C returned by the Histogram Greedy algorithm (Subsection 4.1) and performs the following steps:

1. Check the primitivity of C (using the linear-time algorithm of [7]). If C is primitive, return C.
2. Else, find $V_k \in V_C$ such that if the assignment of V_k is changed from the symbol with the largest value in the row of V_k in $Hist_{V_C,\Sigma}$ to the symbol with the second largest value in this row, thus obtaining a new m-length candidate string C', such that the difference $H(S_L(C'),T) - H(S_L(C),T)$ is minimized and where C' is primitive.

Lemma 18 below describes the time complexity of the Full-Tiling Primitivity Coercion algorithm and immediately follows from the linear-time complexity of the algorithm [7] we use in the first step and the description of the second step.

▶ **Lemma 18.** *The time complexity of the Full-Tiling Primitivity Coercion algorithm is* $O(|\Sigma| \cdot m)$.

Remark: Note that we can use a different algorithm that instead of checking the change of single variables to the second best assignment and choosing the one that gives primitivity with the least number of errors (as our algorithm does), checks the changing to the second best assignment of all subsets of variables and chooses the set that gives primitivity with the least number of errors. This algorithm is obviously correct , i.e., assures primitivity with the least number of errors, however, it has an exponential-time complexity. On the other hand, our algorithm is assured to have polynomial-time complexity, so a proof of its correctness will assure the polynomial-time complexity of the full-tiling relaxation of the ACP.

5.2 Experimental Tests of the Full-Tiling Relaxation Algorithm

Experiment were designed to test the full-tiling relaxation algorithm, which is composed of the algorithms of Subsections 4.1 and 5.1. In particular, we also wanted to experimentally test how many times the full-tiling primitivity coercion is necessary. Note that, due to the result of [3], this algorithm is only of interest to test under a rather high error rate, in which there is an error in every occurrence of the approximate cover of the text, otherwise, the dynamic programming algorithm solving the candidate-relaxation of the ACP is applicable, where trying every substring of T as a candidate cover [3]. In order to comprehensively test the algorithm, the inputs for the tests were classified according to the following criteria:

cover size: A cover C of size m is constructed, where m is small (less than 10), medium (10-100) or large (100-400). Covers of size more than 400 were not created due to space limitations.

alphabet size: The alphabet size was chosen to be either small (at most $\sqrt{m}$) or large (more than $\sqrt{m}$).

tiling style: Given a cover C and its mask M, a tiling L for the text $S_L(C)$ is constructed where the decision of the next index in L is made according to the following styles: random – an equal priority is given to every set bit in M, left priority – a decreasing priority is given to the set bits in M, right priority – an increasing priority is given to the set bits in M.

error rate: The input string T is constructed from $S_L(C)$ by inserting mismatch errors according the following error rates: medium (in every m characters at least one error), high (in every m characters at least $\sqrt{m}$ errors).

error style: The mismatching character is determined according to the following style: random (replacing by a uniformly at random choice of another character from the alphabet) or priority (replacing by another character with priority to the first character in the alphabet, and if the first character is to be replaced then by a uniformly at random chosen different character).

These criteria guarantee that the inputs created for testing the algorithm all have a coverable original string, that its valid tiling is retained. This original string is then introduced with a sufficiently high error rate to produce the current string together with the valid tiling as inputs for the tiling relaxation algorithm. Therefore, all the tested inputs have an L approximate cover and our tiling relaxation algorithm is indeed applicable for them. Moreover, the above criteria for input generation also aim at neutralizing the effect of the cover size, the alphabet size, the tiling style, the error rate or the error style on the validity of the hypothesis, by exhaustively using all reasonable alternatives.

A total of 372000 texts T were constructed as described above and served as inputs (together with the tiling L) to the full-tiling relaxation algorithm. The results are given in Tables 1 and 2 (see Appendix). The column "Percent of Inputs" describes how many of the input texts had each row's characteristics. Numbers are rounded to two digits after decimal point. The column "Identical" describes in how many of the input texts the Histogram Greedy algorithm of Subsection 4.1 returned the original cover C of the text $S_L(C)$ built prior to the error insertion process. The column "Primitive" describes in how many of the input texts the Histogram Greedy algorithm of Subsection 4.1 returned a primitive cover and there was no need to proceed with the second phase of the Full-Tiling Primitivity Coercion algorithm of Subsection 5.1. The column "Non-Primitive" describes in how many of the input texts the Histogram Greedy algorithm of Subsection 4.1 returned a non-primitive string and, therefore, the second phase of the Full-Tiling Primitivity Coercion algorithm of Subsection 5.1 was performed. This latter case happened in 8912 texts, which are about 2% of the texts.

Experiments Conclusion: Primitivity coercion was necessary in 2% of the total tested inputs. In a 100% of the tests the returned string after the Full-Tiling Primitivity Coercion algorithm was indeed an L-approximate cover of the input string.

6 Open Problems

In this paper we initiated the study of the approximate cover problem using a new approach. We proved that the some relaxations (the cover size relaxation) of the approximate cover problem are $\mathcal{NP}$-hard, thus proving that the ACP is $\mathcal{NP}$-hard, while other relaxations (the partial-tiling relaxation and the full-tiling relaxation) are polynomial-time computable. Some interesting questions and open problems are:

- Our $\mathcal{NP}$-hardness proof uses unbounded-size alphabet. Is the ACP still $\mathcal{NP}$-hard for finite alphabet?
- It is interesting to define other relaxations of the ACP and to study their complexity in order to have a deeper understanding of the ACP.
- In this paper we only experimentally checked the correctness of our full-tiling relaxation algorithm. We would like to have a formal proof of its correctness.
- In this paper we considered the Hamming distance as a metric in the definition of approximate cover. Other string metrics can be considered as well. It is interesting to see if and how the complexity of the problem changes with the use of other string metrics.

References

1 Amihood Amir, Estrella Eisenberg, and Avivit Levy. Approximate periodicity. In Otfried Cheong, Kyung-Yong Chwa, and Kunsoo Park, editors, *Proceedings of the 21st International Symposium on Algorithms and Computation (ISAAC 2010)*, volume 6506 of *LNCS*, pages 25–36. Springer, 2010. doi:10.1007/978-3-642-17517-6_5.

2 Amihood Amir, Costas S. Iliopoulos, and Jakub Radoszewski. Two strings at Hamming distance 1 cannot be both quasiperiodic, 2017. arXiv:1703.00195.

3 Amihood Amir, Avivit Levy, Moshe Lewenstein, Ronit Lubin, and Benny Porat. Can we recover the cover? In Juha Kärkkäinen, Jakub Radoszewski, and Wojciech Rytter,

editors, *Proceedings of the 28th Annual Symposium on Combinatorial Pattern Matching (CPM 2017)*, volume 78 of *LIPIcs*, pages 25:1–25:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2017. `doi:10.4230/LIPIcs.CPM.2017.25`.

4 Pavlos Antoniou, Maxime Crochemore, Costas S. Iliopoulos, Inuka Jayasekera, and Gad M. Landau. Conservative string covering of indeterminate strings. In Jan Holub and Jan Zdárek, editors, *Proceedings of the Prague Stringology Conference (PSC 2008)*, pages 108–115. Czech Technical University in Prague, 2008. URL: `http://www.stringology.org/event/2008/p10.html`.

5 Alberto Apostolico and Dany Breslauer. Of periods, quasiperiods, repetitions and covers. In Jan Mycielski, Grzegorz Rozenberg, and Arto Salomaa, editors, *Structures in Logic and Computer Science: A Selection of Essays in Honor of Andrzej Ehrenfeucht*, volume 1261 of *LNCS*, pages 236–248. Springer, 1997. `doi:10.1007/3-540-63246-8_14`.

6 Alberto Apostolico and Andrzej Ehrenfeucht. Efficient detection of quasiperiodicities in strings. *Theor. Comput. Sci.*, 119(2):247–265, 1993. `doi:10.1016/0304-3975(93)90159-Q`.

7 Alberto Apostolico, Martin Farach, and Costas S. Iliopoulos. Optimal superprimitivity testing for strings. *Inf. Process. Lett.*, 39(1):17–20, 1991. `doi:10.1016/0020-0190(91)90056-N`.

8 Dany Breslauer. An on-line string superprimitivity test. *Inf. Process. Lett.*, 44(6):345–347, 1992. `doi:10.1016/0020-0190(92)90111-8`.

9 Dany Breslauer. Testing string superprimitivity in parallel. *Inf. Process. Lett.*, 49(5):235–241, 1994. `doi:10.1016/0020-0190(94)90060-4`.

10 Manolis Christodoulakis, Costas S. Iliopoulos, Kunsoo Park, and Jeong Seop Sim. Approximate seeds of strings. *J. Autom. Lang. Comb.*, 10(5/6):609–626, 2005.

11 Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, Second Edition*, chapter NP-Completeness, pages 966–1021. The MIT Press, 2001.

12 Tim Crawford, Costas S. Iliopoulos, and Rajeev Raman. String-matching techniques for musical similarity and melodic recognition. In Walter B. Hewlett and Eleanor S. Field, editors, *Melodic Similarity: Concepts, Procedures, and Applications*, volume 11 of *Computing in Musicology*, pages 73–100. MIT Press, Cambridge, Massachusetts, 1998.

13 Maxime Crochemore, Costas S. Iliopoulos, Solon P. Pissis, and German Tischler. Cover array string reconstruction. In Amihood Amir and Laxmi Parida, editors, *Proceedings of the 21st Annual Symposium on Combinatorial Pattern Matching (CPM 2010)*, volume 6129 of *LNCS*, pages 251–259. Springer, 2010. `doi:10.1007/978-3-642-13509-5_23`.

14 Maxime Crochemore, Costas S. Iliopoulos, and Hiafeng Yu. Algorithms for computing evolutionary chains in molecular and musical sequences. In Costas S. Iliopoulos, editor, *Proceedings of the 9th Australian Workshop on Combinatorial Algorithms (AWOCA 1998)*, pages 172–184, France, 1998. URL: `https://hal-upec-upem.archives-ouvertes.fr/hal-00619988/`.

15 Tomás Flouri, Costas S. Iliopoulos, Tomasz Kociumaka, Solon P. Pissis, Simon J. Puglisi, William F. Smyth, and Wojciech Tyczyński. Enhanced string covering. *Theor. Comput. Sci.*, 506:102–114, 2013. `doi:10.1016/j.tcs.2013.08.013`.

16 Ondřej Guth and Bořivoj Melichar. Using finite automata approach for searching approximate seeds of strings. In Xu Huang, Sio-Iong Ao, and Oscar Castillo, editors, *Intelligent Automation and Computer Engineering*, volume 52 of *Lecture Notes in Electrical Engineering*, pages 347–360. Springer Netherlands, 2010. `doi:10.1007/978-90-481-3517-2_27`.

17 Ondřej Guth, Bořivoj Melichar, and Miroslav Balík. Searching all approximate covers and their distance using finite automata. In Peter Vojtáš, editor, *Proceedings of the Conference on Theory and Practice of Information Technologies (ITAT 2008)*, volume 414 of *CEUR*

Workshop Proceedings, pages 21–26, 2009. URL: `http://ceur-ws.org/Vol-414/paper4.pdf`.

18 Costas S. Iliopoulos, Dennis W. G. Moore, and Kunsoo Park. Covering a string. *Algorithmica*, 16(3):288–297, 1996. doi:10.1007/BF01955677.

19 Costas S. Iliopoulos and Laurent Mouchard. Quasiperiodicity and string covering. *Theor. Comput. Sci.*, 218(1):205–216, 1999. doi:10.1016/S0304-3975(98)00260-6.

20 Costas S. Iliopoulos and William F. Smyth. An on-line algorithm of computing a minimum set of k-covers of a string. In Costas S. Iliopoulos, editor, *Proceedings of the 9th Australian Workshop on Combinatorial Algorithms (AWOCA 1998)*, 1998.

21 Tomasz Kociumaka, Solon P. Pissis, Jakub Radoszewski, Wojciech Rytter, and Tomasz Waleń. Fast algorithm for partial covers in words. *Algorithmica*, 73(1):217–233, 2015. doi:10.1007/s00453-014-9915-3.

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

23 Gad M. Landau and Jeanette P. Schmidt. An algorithm for approximate tandem repeats. In Alberto Apostolico, Maxime Crochemore, Zvi Galil, and Udi Manber, editors, *Proceedings of the 4th Annual Symposium on Combinatorial Pattern Matching (CPM 1993)*, volume 684 of *LNCS*, pages 120–133. Springer, 1993. doi:10.1007/BFb0029801.

24 Gad M. Landau, Jeanette P. Schmidt, and Dina Sokol. An algorithm for approximate tandem repeats. *J. Comput. Biol.*, 8(1):1–18, 2001. doi:10.1089/106652701300099038.

25 Yin Li and William F. Smyth. Computing the cover array in linear time. *Algorithmica*, 32(1):95–106, 2002. doi:10.1007/s00453-001-0062-2.

26 M. Lothaire, editor. *Combinatorics on words.* Cambridge Mathematical Library. Cambridge University Press, 1997. doi:10.1017/CBO9780511566097.

27 Dennis Moore and William F. Smyth. An optimal algorithm to compute all the covers of a string. *Inf. Process. Lett.*, 50(5):239–246, 1994. doi:10.1016/0020-0190(94)00045-X.

28 Dennis Moore and William F. Smyth. A correction to "An optimal algorithm to compute all the covers of a string". *Inf. Process. Lett.*, 54(2):101–103, 1995. doi:10.1016/0020-0190(94)00235-Q.

29 Jeong Seop Sim, Costas S. Iliopoulos, Kunsoo Park, and William F. Smyth. Approximate periods of strings. *Theor. Comput. Sci.*, 262(1):557–568, 2001. doi:10.1016/S0304-3975(00)00365-0.

30 William F. Smyth. Repetitive perhaps, but certainly not boring. *Theor. Comput. Sci.*, 249(2):343–355, 2000. doi:10.1016/S0304-3975(00)00067-0.

31 Hui Zhang, Qing Guo, and Costas S. Iliopoulos. Algorithms for computing the lambda-regularities in strings. *Fundam. Inform.*, 84(1):33–49, 2008. URL: `http://content.iospress.com/articles/fundamenta-informaticae/fi84-1-04`.

32 Hui Zhang, Qing Guo, and Costas S. Iliopoulos. Varieties of regularities in weighted sequences. In Bo Chen, editor, *Proceedings of the 6th International Conference on Algorithmic Aspects in Information and Management (AAIM 2010)*, volume 6124 of *LNCS*, pages 271–280. Springer, 2010. doi:10.1007/978-3-642-14355-7_28.

Table 1 Experimental Tests of the Full-Tiling Relaxation Algorithm for Small Alphabets.

Cover Size	Tiling Style	Error Rate	Error Style	Percent of Inputs	Identical	Primitive	Non-Primitive
small	left	medium	random	1.57	80.89	13.04	6.07
small	left	medium	priority	1.57	80.55	31.51	5.95
small	random	medium	random	1.57	78.80	15.28	5.93
small	random	medium	priority	1.57	78.30	15.46	6.24
small	right	medium	random	1.57	76.20	17.47	6.32
small	right	medium	priority	1.57	76.13	17.82	6.05
small	left	high	random	1.57	5.55	77.78	16.67
small	left	high	priority	1.57	1.87	81.39	16.74
small	random	high	random	1.57	5.24	78.57	16.19
small	random	high	priority	1.57	1.68	82.18	16.13
small	right	high	random	1.57	4.74	80.41	14.85
small	right	high	priority	1.57	1.27	84.28	14.45
medium	left	medium	random	3.22	100	0	0
medium	left	medium	priority	3.22	100	0	0
medium	random	medium	random	3.22	100	0	0
medium	random	medium	priority	3.22	100	0	0
medium	right	medium	random	3.22	100	0	0
medium	right	medium	priority	3.22	100	0	0
medium	left	high	random	3.22	89.80	10.17	0.03
medium	left	high	priority	3.22	87.87	12.09	0.03
medium	random	high	random	3.22	89.39	10.59	0.02
medium	random	high	priority	3.22	87.42	12.54	0.05
medium	right	high	random	3.22	89.07	10.90	0.33
medium	right	high	priority	3.22	86.63	13.35	0.03
large	left	medium	random	0.81	100	0	0
large	left	medium	priority	0.81	100	0	0
large	random	medium	random	0.81	100	0	0
large	random	medium	priority	0.81	100	0	0
large	right	medium	random	0.81	100	0	0
large	right	medium	priority	0.81	100	0	0
large	left	high	random	0.81	100	0	0
large	left	high	priority	0.81	100	0	0
large	random	high	random	0.81	100	0	0
large	random	high	priority	0.81	100	0	0
large	right	high	random	0.81	100	0	0
large	right	high	priority	0.81	100	0	0

Table 2 Experimental Tests of the Full-Tiling Relaxation Algorithm for Large Alphabets.

Cover Size	Tiling Style	Error Rate	Error Style	Percent of Inputs	Identical	Primitive	Non-Primitive
small	left	medium	random	0.59	89.45	9.59	0.96
small	left	medium	priority	0.59	76.51	21.93	1.56
small	random	medium	random	0.59	87.57	11.24	1.19
small	random	medium	priority	0.59	76.15	22.39	1.47
small	right	medium	random	0.59	86.56	12.89	0.55
small	right	medium	priority	0.59	75.51	23.40	1.10
small	left	high	random	0.59	25.55	70.78	3.67
small	left	high	priority	0.59	1.84	84.45	13.72
small	random	high	random	0.59	24.82	70.96	4.22
small	random	high	priority	0.59	2.06	86.15	11.79
small	right	high	random	0.59	23.76	72.75	3.49
small	right	high	priority	0.59	1.88	85.32	12.80
medium	left	medium	random	1.62	100	0	0
medium	left	medium	priority	1.62	100	0	0
medium	random	medium	random	1.62	100	0	0
medium	random	medium	priority	1.62	100	0	0
medium	right	medium	random	1.62	100	0	0
medium	right	medium	priority	1.62	100	0	0
medium	left	high	random	1.62	99.77	0.23	0
medium	left	high	priority	1.62	85.11	14.89	0
medium	random	high	random	1.62	99.75	0.25	0
medium	random	high	priority	1.62	84.19	15.81	0
medium	right	high	random	1.62	99.90	0.10	0
medium	right	high	priority	1.62	84.11	15.90	0
large	left	medium	random	0.54	100	0	0
large	left	medium	priority	0.54	100	0	0
large	random	medium	random	0.54	100	0	0
large	random	medium	priority	0.54	100	0	0
large	right	medium	random	0.54	100	0	0
large	right	medium	priority	0.54	100	0	0
large	left	high	random	0.54	100	0	0
large	left	high	priority	0.54	100	0	0
large	random	high	random	0.54	100	0	0
large	random	high	priority	0.54	100	0	0
large	right	high	random	0.54	100	0	0
large	right	high	priority	0.54	100	0	0