Cadences in Grammar-Compressed Strings

Julian Pape-Lange(✉)

Technische Universität Chemnitz, Straße der Nationen 62, 09111 Chemnitz, Germany
julian.pape-lange@informatik.tu-chemnitz.de

Abstract. Cadences are structurally maximal arithmetic progressions of indices corresponding to equal characters in an underlying string.

This paper provides a detection algorithm for 3-cadences in binary strings which runs in linear time on uncompressed strings and in polynomial time on grammar-compressed strings.

Furthermore, this paper proves that several variants of the cadence detection problem are $\mathcal{NP}$-complete on grammar-compressed strings and that the equidistant subsequence matching problem with patterns of length three is $\mathcal{NP}$-complete on grammar-compressed ternary strings.

Keywords: String-cadences · String algorithms · Compressed pattern matching

1 Introduction

A sub-cadence in a string is an arithmetic progression of indices corresponding to equal characters. Van der Waerden shows in [12] that for each k and each alphabet size $|\Sigma|$, there is a natural number $m(k, |\Sigma|)$, such that each sequence of characters in Σ with length greater than or equal to $m(k, |\Sigma|)$ has a sub-cadence consisting of k indices. However, the term cadence in the context of strings was first used by Gardelle in [5] in the year 1964.

In this paper, we use the notation of Amir et al. in [1] and say that a cadence is a sub-cadence which is structurally maximal in the sense that the extension of the arithmetic progression to the left or to the right would not result in a valid index of the string.

For example, in the string $S = 10101$, the three indices $1, 3, 5$ form a cadence, since the indices -1 and 7 are both outside of the string. On the other hand, in the string $S = 01110$, the three indices $2, 3, 4$ do not form a cadence, since, for example, the index 1 is inside the string.

Funakoshi and Pape-Lange prove in [4] that if the underlying alphabet has a constant size, the number of 3-cadences in an uncompressed string of length n can be counted in $\mathcal{O}(n(\log n)^2)$ time using fast Fourier transform.

Furthermore, Funakoshi et al. present in [3] the more general problem of equidistant subsequence matching which extends the sub-cadences to arbitrary arithmetic factors, and showed that techniques for cadence-detection can be adopted to solve equidistant subsequence matching with similar time complexity.

A. Leporati et al. (Eds.): LATA 2021, LNCS 12638, pp. 331–342, 2021.
https://doi.org/10.1007/978-3-030-68195-1_26

Strings can be compressed by straight-line programs, which are context-free grammars whose languages contain exactly one string each. Since this grammar-based compression is able to compress some strings to logarithmic size, we are interested which polynomial time problems on uncompressed strings can also be solved in polynomial time with respect to the compressed size of the string. For example, grammar-based compression allows for fast algorithms as the fully compressed pattern matching by Jeż presented in [6]. Also, the size of the smallest grammar is comparable to other strong string compression algorithms as LZ77 (as proven simultaneously by Rytter in [11] and by Charikar et al. in [2]) and hence as the run-length encoded Burrows-Wheeler transform (as recently proven independently by Kempa and Kociumaka in [7] and by Pape-Lange in [10]).

In this paper, we will consider several variants of the k-cadence detection problem. I.e. the decision problem on whether such a k-cadence occurs in a given string.

We prove that the 3-cadence detection problem can be solved in linear time on an uncompressed binary string and in polynomial time on a grammar-compressed binary string.

Furthermore, on grammar-compressed strings, the cadence detection problem becomes $\mathcal{NP}$-complete for longer cadences or 3-cadences over a ternary alphabet.

In order to obtain these algorithms, this paper introduces two new special cases of the sub-cadence, the L-R-cadence, which starts and ends in given intervals, and the even/odd 3-sub-cadence, which starts at even/odd indices.

2 Preliminaries

A string S of length n is the concatenation $S = S[1]S[2]S[3]\dots S[n]$ of characters from an alphabet Σ. Strings naturally split into runs of equal characters. For example, the string 00010101100 splits into $000 \cdot 1 \cdot 0 \cdot 1 \cdot 0 \cdot 11 \cdot 00$. In this paper, these runs of equal characters are just called runs for the sake of simplicity.

For the sub-cadences and cadences, this paper uses the definitions of Amir et al. in [1]. These definitions are slightly different from the definition by Gardelle in [5] and by Lothaire in [9]. Funakoshi and Pape-Lange present a comparison of these definitions in [4].

Definition 1. *A* k-sub-cadence *is an arithmetic progression*

$$(i, i+d, \dots, i+(k-1)d)$$

of indices given by the triple (i, d, k) *of integers with* $d, k > 0$ *such that*

$$S[i] = S[i+d] = \dots = S[i+(k-1)d]$$

holds.

As a special case, cadences additionally have to be *structurally maximal* in the sense that neither of the extensions of the underlying arithmetic progression is contained in the integer interval $\{1, 2, 3, \dots, n\}$ anymore. More formally:

Definition 2. *A* k-cadence *is a k-sub-cadence (i, d, k) such that the inequalities $i - d \leq 0$ and $n < i + kd$ hold.*

In this paper, we will also consider a new special case of the sub-cadence, in which the first element and the last element of the sub-cadence are contained in given intervals:

Definition 3. *For two disjoint intervals L and R, an* L-R-k-cadence *is a k-sub-cadence (i, d, k) which starts in the interval L and ends in the interval R. I.e. $i \in L$ and $i + (k - 1)d \in R$ hold.*

Since the first element and the third element of each 3-sub-cadence have the same parity, it is useful to divide the L-R-3-cadences and 3-cadences according to this parity. Without loss of generality, we will only consider the even sub-cadences.

Definition 4. *An L-R-3-cadence/3-cadence is* even *if its first element is an even number and* odd *otherwise.*

For each set M, we define $M_{\text{even}} := M \cap 2\mathbb{Z}$ and $M_{\text{odd}} := M \cap (2\mathbb{Z} + 1)$ and for each $M = \{a_1, a_2, \ldots, a_l\} \subset \mathbb{Z}$ with $1 \leq a_1 < a_2 < a_3 < \cdots < a_l \leq n$, we define the string $S[M] = S[a_1]S[a_2] \ldots S[a_l]$ as the subsequence of characters with indices given by M.

The string S_{even} is defined by $S_{\text{even}} := S\left[\left\{2, 4, 6, \ldots, 2\left\lfloor \frac{|S|}{2} \right\rfloor\right\}\right]$.

For the compressed problems, we consider the strings to be given by straight-line grammars.

Definition 5. *A* straight-line grammar *is a context-free grammar (V, Σ, R, S) with variables $V = \{v_1, v_2, \ldots, v_i\}$ such that for each variable v_i there is exactly one rule $v_i \to u_1 u_2 \ldots u_j$ and each u_k on the right-hand side is either a character in Σ or a variable $v_{k'}$ in V with a smaller index than v_i. I.e. $k' < i$.*

The size *of a straight-line grammar is given by the total length of the right-hand sides of the rules.*

These straight-line grammars allow on the one hand compression to logarithmic size and on the other hand fully compressed pattern matching in polynomial time with respect to the compressed sizes of the string and the pattern.

3 NP-Complete Cadence Problems

In this section, we will prove the following theorem:

Theorem 1. *If at least one of the following conditions holds, the k-cadence detection problem on compressed strings is $\mathcal{NP}$-complete:*

- *$k \geq 3$ and $|\Sigma| \geq 2$ and we only consider k-cadences with a given character,*
- *$k \geq 3$ and $|\Sigma| \geq 3$ or*
- *$k \geq 4$ and $|\Sigma| \geq 2$.*

Since we can test for a given candidate (i, d, k) of a k-cadence in polynomial time, whether (i, d, k) forms indeed a k-cadence, all three problems mentioned above belong to $\mathcal{NP}$ and it is left to show that they are $\mathcal{NP}$-hard.

To show the $\mathcal{NP}$-hardness, we will reduce the following problem, which Lohrey proves in Theorem 3.13 of [8] to be $\mathcal{NP}$-complete, to the problems above:

input: Two compressed strings P and P' over the alphabet $\{0, 1\}$.
output: Is there an index l with $P[l] = P'[l] = 1$?

Let P and P' be compressed strings over the alphabet $\{0, 1\}$. Without loss of generality, the inequality $|P'| \leq |P|$ holds. Define $P'' = (P'0^{|P|-|P'|})_{\text{rev}}$.

In this setting, for every index l, the equation $P[l] = P'[l] = 1$ holds if and only if the equation $P[l] = P''[|P| + 1 - l] = 1$ holds as well.

Consider the string

$$S = \left(0^{(k-1)|P|} \cdot P \cdot 0 \cdot 0^{k|P|}\right)\left(0^{k|P|} \cdot 1 \cdot 0^{k|P|}\right)\left(0^{k|P|} \cdot 0 \cdot P'' \cdot 0^{(k-1)|P|}\right)\left(1^{2k|P|+1}\right)^{k-3},$$

A grammar of this string can be built by the grammars of P and P' and $\mathcal{O}\left(\log(k^2|P|)\right)$ additional nonterminals. Since the compression of a string P needs at least $\Omega(\log|P|)$ nonterminals, the compressed size of S is, for fixed k, polynomial in the compressed size of the inputs.

If there is an index l with $P[l] = P''[|P| + 1 - l] = 1$, the corresponding indices are contained in the arithmetic progression starting at $i = (k-1)|P| + l$ with distance $d = 2k|P| + 1 + (|P| + 1 - l)$ and length k.

For each $-1 \leq j \leq k$ the inequality $j(2k|P|+1) < i + jd \leq (j+1)(2k|P|+1)$ holds. Therefore, the indices of the arithmetic progression starting at the index $i = (k-1)|P| + l$ with distance $d = 2k|P| + 1 + (|P| + 1 - l)$ and length k correspond to 1s in S and the inequalities $i - d \leq 0$, $i > 0$, $i + (k-1)d \leq n$ and $i + kd > n$ hold. Therefore, this arithmetic progression is a k-cadence.

Conversely, if the triple (i, d, k) defines a k-cadence with character 1 in S, the inequalities $i - d \leq 0 < i$ and $i + (k-1)d \leq n < i + kd$ of the cadence imply

$$\frac{j}{k}n < \frac{k-j}{k}i + \frac{j}{k}(i+kd) = i + jd = \frac{k-j-1}{k}(i-d) + \frac{j+1}{k}(i+(k-1)d) \leq \frac{j+1}{k}n.$$

In particular, the index $i + d$ has to be the single 1 in the second bracket which has the index $(2k|P| + 1) + (k|P|) + 1$. Furthermore, the first element of the k-cadence has to be a 1 in P in the first bracket and the third element of the k-cadence has to be a 1 in P'' in the third bracket.

By construction, the two indices of these characters have the same distance to the index $(2k|P| + 1) + (k|P|) + 1$, and the two strings P and P'' have the same distance to the index $(2k|P| + 1) + (k|P|) + 1$ as well. Therefore, the first element of the k-cadence and the third element of the k-cadence define an index l with $P[l] = P''[|P| + 1 - l] = 1$.

Therefore, the string S has a k-cadence with character 1 if and only if there is an index l such that $P[l] = P'[l] = 1$ holds.

If $k > 3$ holds, the requirement that the underlying character has to be 1 can be dropped since there is at least one bracket in S containing only 1s.

For 3-cadences on a ternary alphabet we consider the string

$$S = \left(0^{(k-1)|P|} \cdot P \cdot 0 \cdot 0^{k|P|}\right)\left(2^{k|P|} \cdot 1 \cdot 2^{k|P|}\right)\left(0^{k|P|} \cdot 0 \cdot P'' \cdot 0^{(k-1)|P|}\right)$$

which similarly has a 3-cadence if and only if there is an index l such that $P[l] = P'[l] = 1$ holds.

This concludes the proof of Theorem 1.

4 L-R-Cadences

The algorithm of Funakoshi and Pape-Lange in [4] counts the 3-cadences of an uncompressed string of length n in $\mathcal{O}\left(n(\log n)^2\right)$ time. This is done by counting the L-R-3-cadences in $\mathcal{O}\left((|L| + |R|)(\log(|L| + |R|))\right)$ time. It therefore seems reasonable to understand the L-R-cadences to be a simplification of cadences.

Nevertheless, in this section, we will show that all detection problems on compressed strings discussed in the last section are also $\mathcal{NP}$-complete for the L-R-cadences, even if $k = 3$ and $|\Sigma| = 2$ hold.

However, for $k = 3$ and $|\Sigma| = 2$, we will provide a detection algorithm for L-R-3-cadences which needs on uncompressed strings only $\mathcal{O}\left(|L| + |R|\right)$ time and on compressed strings polynomial time with respect to the compressed size of the string and the additional variable $\max\left(\frac{|L|}{|R|}, \frac{|R|}{|L|}\right)$.

Theorem 2. *For $k \geq 3$ and $|\Sigma| \geq 2$, the L-R-k-cadence detection problem is $\mathcal{NP}$-complete on compressed strings.*

Proof. If either $k > 3$ or $|\Sigma| > 2$ hold or if we only consider L-R-k-cadences with a given character, the proof is essentially equal to the corresponding proof in the last section, since for $L = \left\{1, 2, \ldots, \frac{1}{k}n\right\}$ and $R = \left\{\frac{k-1}{k}n + 1, \frac{k-1}{k}n + 2, \ldots, n\right\}$, all k-cadences in the discussed string S are L-R-k-cadences and vice versa.

Otherwise, we have $k = 3$, $|\Sigma| = 2$ and we consider L-R-3-cadences with any character. Since we can test for every triple (i, d, k), whether this triple forms an L-R-3-cadence, this problem belongs to $\mathcal{NP}$ and it is left to show that even this special case is $\mathcal{NP}$-hard.

Let P and P' be compressed strings over the alphabet $\{0, 1\}$. Without loss of generality, the inequality $|P'| \leq |P|$ holds. Define P'' to be the string which results from duplicating all characters of P'. For example, for $P' = 011$, we define $P'' = 001111$. This can be done by introducing two additional nonterminals.

Consider the string $S = 1(0^{|P|})(P)(P'')$ as well as the intervals $L = \{1\}$ and $R = \{1 + 2|P| + 1, 1 + 2|P| + 2, \ldots 1 + 2|P| + |P''|\}$. In this setting $S[L] = 1$ and $S[R] = P''$ holds. Furthermore, for each index $1 \leq l \leq |P|$, the equations $P[l] = S[1 + (|P| + l)]$ and $P'[l] = P''[2l] = S[1 + 2|P| + 2l] = S[1 + 2(|P| + l)]$ hold.

Therefore, for each index l, the equation $P[l] = 1 = P'[l]$ holds if and only if the equation $S[1] = S[1 + (|P| + l)] = S[1 + 2(|P| + l)]$ holds. This equation, however, defines an L-R-3-cadence.

This concludes the proof of Theorem 2.

Similarly, for two compressed strings P over $\{0,1\}$ and P' over $\{0,2\}$, we can define P'' as above and the string $S = 2(0^{|P|})(P)(P'')$ has an equidistant occurrence of the pattern 212 if and only if there is an index i with $P[i] = 1$ and $P'[i] = 2$. Therefore, equidistant subsequence matching with patterns of length 3 on compressed ternary strings is also $\mathcal{NP}$-complete.

All reductions above used that we could force all cadences to use a fixed character of the string. However, surprisingly, if L and R have similar length, we can detect in polynomial time, whether a compressed binary string has an L-R-3-cadence. Furthermore, with the same idea we can detect in $\mathcal{O}(|L| + |R|)$ time, whether an uncompressed binary string has an L-R-3-cadence.

The remainder of this section will prove the following theorem:

Theorem 3. *The problem of L-R-3-cadence detection can be done on uncompressed binary strings in $\mathcal{O}(|L| + |R|)$ time and on compressed binary strings in polynomial time with respect to the compressed size of the string and the additional variable* $\max\left(\frac{|L|}{|R|}, \frac{|R|}{|L|}\right)$.

The key insight for the detection algorithm for L-R-3-cadences is that if the string does not contain L-R-3-cadences, either $S[L_{\text{even}}]$ or $S[R_{\text{even}}]$ is very structured. The following lemma implies that if $S[L_{\text{even}}]$ has the substring 01 and $S[R_{\text{even}}]$ has the substring 10 or vice versa, then S has an L-R-3-cadence:

Lemma 1. *Let S be a binary string and L and R be two intervals.*
If there are indices i and j with

- $S[i] = S[j] \neq S[i+2] = S[j-2]$,
- $i, i+2 \in L$,
- $j, j-2 \in R$ *and*
- $i \equiv j \pmod 2$,

then S has an L-R-3-cadence.

Proof. Since $i \equiv j \pmod 2$ holds, the number $\frac{i+j}{2}$ is an integer. Furthermore, since S is binary and $S[i] = S[j] \neq S[i+2] = S[j-2]$ holds, we either have $S[i] = S[\frac{i+j}{2}] = S[j]$ or $S[i+2] = S[\frac{i+j}{2}] = S[j-2]$. Therefore, there is at least one L-R-3-cadence.

This implies that if S does not contain L-R-3-cadences, then there are only few possibilities for the subsequences $S[L_{\text{even}}]$ and $S[R_{\text{even}}]$:

Corollary 1. *Let S be a binary string and L and R be two intervals such that S has no L-R-3-cadences.*
Then, either

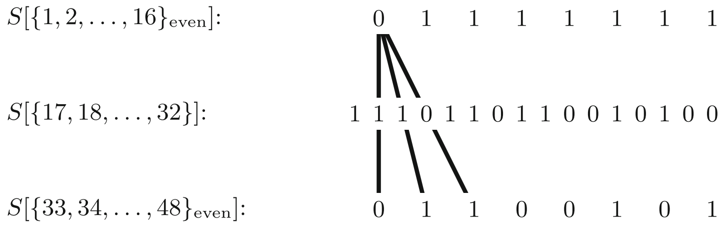
- $S[L_{\text{even}}]$ *or* $S[R_{\text{even}}]$ *is of the form* 0^j *or* 1^j,
- *both* $S[L_{\text{even}}]$ *or* $S[R_{\text{even}}]$ *are of the form* 0^i1^j *or*
- *both* $S[L_{\text{even}}]$ *or* $S[R_{\text{even}}]$ *are of the form* 1^i0^j.

If both $S[L_{\text{even}}]$ and $S[R_{\text{even}}]$ are of the form 0^i1^j or 1^i0^j, we can divide L and R into intervals L', L'', R' and R'' such that $S[L'_{\text{even}}]$ and $S[R'_{\text{even}}]$ are of the form 0^i and $S[L''_{\text{even}}]$ and $S[R''_{\text{even}}]$ are of the form 1^i. This can be done in $\mathcal{O}(|L| + |R|)$ time on uncompressed strings and in polynomial time on compressed strings.

By construction, all even L-R-3-cadences are either even L'-R'-3-cadences or even L''-R''-3-cadences. The following lemma, which holds by definition of the even L-R-3-cadence, shows that they can be detected in $\mathcal{O}(|L| + |R|)$ time in uncompressed strings and in polynomial time in compressed strings.

Lemma 2. *Let S be a binary string and L and R be two intervals such that $S[L_{\text{even}}] = 0^i$ and $S[R_{\text{even}}] = 0^j$ hold. Let further $l_{\min}$, $l_{\max}$, $r_{\min}$ and $r_{\max}$ be the minimal and maximal indices of L_{even} and R_{even}, respectively.*

Then, the string $S\left[\left\{\frac{l_{\min}+r_{\min}}{2}, \frac{l_{\min}+r_{\min}}{2}+1, \ldots, \frac{l_{\max}+r_{\max}}{2}\right\}\right]$ contains a 0 if and only if S has an even L-R-3-cadence.



$S[\{1,2,\ldots,16\}_{\text{even}}]$:
0 1 1 1 1 1 1 1
$S[\{17,18,\ldots,32\}]$:
1 1 1 0 1 1 0 1 1 0 0 1 0 1 0 0
$S[\{33,34,\ldots,48\}_{\text{even}}]$:
0 1 1 0 0 1 0 1

Fig. 1. A string with 48 characters. For $L = \{2\}$ and $R = \{33, 34, \ldots, 48\}$, for each index of R_{even}, there is only one candidate (i, d, k) for forming an L-R-3-cadence.

The more difficult case is that one of the two subsequences contains the substrings 01 and 10 while the other subsequence contains neither of these two substrings. Figure 1 shows that if L is a short interval, we may have to check linearly many pairs with respect to the length of the string in order to find an L-R-3-cadence. This case occurred, for example, in the string $S = 1(0^{|P|})(P)(P'')$ in which the L-R-3-cadence detection was $\mathcal{NP}$-hard. However, it turns out that even in this case, the detection of L-R-3-cadences can be done in $\mathcal{O}(|L| + |R|)$ time on uncompressed strings and using the additional variable $\max\left(\frac{|L|}{|R|}, \frac{|R|}{|L|}\right)$, the detection can also be done in polynomial time on compressed strings.

By definition of the L-R-3-cadence we get the following lemma:

Lemma 3. *Let S be a string and L and R be two intervals. Let further $S[L_{\text{even}}]$ be of the form 0^i and $l_{\min}$, $l_{\max}$, $r_{\min}$ and $r_{\max}$ be the minimal and maximal indices of L_{even} and R_{even}, respectively.*

Then, for any $r_0 \in R_{\text{even}}$ with $S[r_0] = 0$, there is an even L-R-3-cadence which uses r_0 as last element if and only if $S\left[\left\{\frac{l_{\min}+r_0}{2}, \frac{l_{\min}+r_0}{2}+1, \ldots, \frac{l_{\max}+r_0}{2}\right\}\right]$ contains a 0.

Also, for any $m_0 \in \left\{\frac{l_{\min}+r_{\min}}{2}, \frac{l_{\min}+r_{\min}}{2}+1, \ldots, \frac{l_{\max}+r_{\max}}{2}\right\}$ *with* $S[m_0]=0$, *define* $r'_{\min} = \max\left(2m_0 - l_{\max}, r_{\min}\right)$ *and* $r'_{\max} = \min\left(2m_0 - l_{\min}, r_{\max}\right)$. *There is an even* L-R-3*-cadence which uses this* 0 *as middle element if and only if* $S\left[\left\{r'_{\min}, r'_{\min}+2, \ldots, r'_{\max}\right\}\right]$ *contains a* 0.

With Corollary 1, Lemma 2 and Lemma 3, it is possible to efficiently either find an L-R-3-cadence or to shorten R without removing any L-R-3-cadences. The resulting algorithm is also presented in Fig. 2.

Corollary 2. *Let* S *be a binary string and* L *and* R *be two intervals. Let further* $S[L_{\text{even}}]$ *be of the form* 0^i *and* $l_{\min}$, $l_{\max}$, $r_{\min}$ *and* $r_{\max}$ *be the minimal and maximal indices of* L_{even} *and* R_{even}, *respectively.*

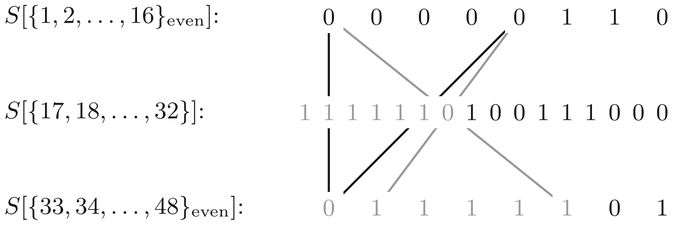
If $S[R_{\text{even}}]$ *is of the form* 1^j, *there is no even* L-R-3*-cadence.*

Otherwise, define $r_0 = \min\left\{r \in R_{\text{even}} \middle| S[r]=0\right\}$ *and use Lemma 3 to check whether there is an* L-R-3*-cadence using an index of* L_{even} *and* r_0.

If such an L-R-3*-cadence does not exist, define* $m_{\min} = \frac{l_{\max}+r_0}{2}+1$ *and* $m_{\max} = \frac{l_{\max}+r_{\max}}{2}$. *If the substring* $S\left[\left\{m_{\min}, m_{\min}+1, \ldots, m_{\max}\right\}\right]$ *of* S *is of the form* 1^j, *then there is no even* L-R-3*-cadence.*

Otherwise, define $m_0 = \min\left\{m \in \left\{m_{\min}, m_{\min}+1, \ldots, m_{\max}\right\} \middle| S[m]=0\right\}$. *Then use Lemma 3 to check whether there is an* L-R-3*-cadence using an index of* L_{even} *and* m_0.

If such an L-R-3*-cadence does not exist, define* $R' = R \cap \mathbb{Z}_{>2m_0-l_{\min}}$. *There is an even* L-R-3*-cadence if and only if there is an even* L-R'-3*-cadence.*



$S[\{1,2,\ldots,16\}_{\text{even}}]$:
0 0 0 0 0 1 1 0
$S[\{17,18,\ldots,32\}]$:
1 1 1 1 1 1 0 1 0 0 1 1 1 0 0 0
$S[\{33,34,\ldots,48\}_{\text{even}}]$:
0 1 1 1 1 1 0 1

Fig. 2. A string with 48 characters after one application of Corollary 2. Let the intervals $L=\{2,3,\ldots,10\}$ and $R=\{33,34,\ldots,48\}$ be given. First, the index $r_0=34$ is found. The minimal and maximal candidates for 3-cadences with r_0 are given by the black lines. Then, the index $m_0=23$ is found. The minimal and maximal candidates for 3-cadences with m_0 are given by the gray lines. Afterwards, the gray characters are guaranteed not to form an L-R-3-cadence with characters from the first run of the string.

In the uncompressed case, iterated application of Corollary 2 reads each character of $S\left[\left\{\frac{l_{\min}+r_{\min}}{2}, \frac{l_{\min}+r_{\min}}{2}+1, \ldots, \frac{l_{\max}+r_{\max}}{2}\right\}\right]$ and $S[R_{\text{even}}]$ at most once. It is therefore easy to see that iterated application of this corollary only uses $\mathcal{O}\left(|L|+|R|\right)$ time to decide, whether an L-R-3-cadence exists.

On the other hand, after the application of Corollary 2 either R' is empty or it contains at least $|L|$ elements less than R. Therefore, the algorithm described in Corollary 2 has to be used at most $\mathcal{O}\left(\frac{|R|}{|L|}\right)$ times. Since each application of Corollary 2 only takes polynomial time on compressed strings, the L-R-3-cadence detection on compressed strings can be done in polynomial time with respect to the compressed size of the string and the additional variable $\max\left(\frac{|L|}{|R|}, \frac{|R|}{|L|}\right)$.

Since by symmetry, the detection of odd L-R-3-cadences can also be done as the detection of even L-R-3-cadences, this concludes the proof of Theorem 3.

5 3-Cadences in Binary Strings

In this section, we will show that the results of Theorem 3 also hold for the corresponding 3-cadence problems:

Theorem 4. *The 3-cadence detection problem can be solved in linear time on uncompressed binary strings and in polynomial time on compressed binary strings.*

Let i, d be two integers such that $i - d \leq 0$ and $i + 3d > n$ hold. Let $L = \{1, 2, \dots, i\}$ and $R = \{i + 2d, i + 2d + 1, \dots, n\}$ be two intervals. Then each L-R-3-cadence is also a 3-cadence. On the other hand, each 3-cadence defines integers i and d such that $i - d \leq 0$ and $i + 3d > n$ hold. Therefore, Lemma 1 implies that if S has an even 3-cadence, it also has a 3-cadence that either starts in one of the first two runs of S_{even} or ends in one of the last two runs of S_{even}.

The main challenge for the adaption of the detection algorithm for L-R-3-cadences to an detection algorithm for 3-cadences is that while each character in the first/last third of the string can be the first/last index in a 3-cadence, not all 3-sub-cadences which start in the first third and end in the last third are 3-cadences. For example, in the string 001001001 the three 1s form a 3-cadence while the in the string 001010100 the three 1s do not form a 3-cadence. Therefore, Lemma 3 does not quite work on 3-cadences and we have to restrict the strings in Lemma 3 to those indices such that the corresponding 3-sub-cadences are structurally maximal.

The following lemma restricts the bound of Lemma 3 to the allowed indices for 3-cadences with $i - d \leq 0$ and $i + kd > n$ and therefore holds by definition of the 3-cadence:

Lemma 4. *Let S be a string. Let further S_{even} be of the form $0^i S'$ and let $l_{\min} = 2$ and $l_{\max} = \max\left(2i, 2\left\lfloor \frac{1}{6} n \right\rfloor\right)$ be the first and the last indices of the first run of S_{even} which can be the first index of a 3-cadence, respectively.*

Then, for any $r_0 \in S_{\text{even}}$ define $l'_{\max} = \min\left(l_{\max}, 2\left\lfloor \frac{r_0}{6} \right\rfloor, 2(\left\lceil \frac{3r_0 - 2n}{2} \right\rceil - 1)\right)$. There is an even 3-cadence which uses this r_0 as last element and any of the first i 0s of $S[L_{\text{even}}]$ as first element if and only if $S[r_0] = 0$ holds and the string $S\left[\left\{\frac{l_{\min} + r_0}{2}, \frac{l_{\min} + r_0}{2} + 1, \dots, \frac{l'_{\max} + r_0}{2}\right\}\right]$ contains a 0.

Conversely, for any $m_0 \in \left\{\frac{l_{\min}+2\left\lfloor\frac{n}{3}\right\rfloor+2}{2}, \frac{l_{\min}+2\left\lfloor\frac{n}{3}\right\rfloor+2}{2}+1, \dots, \frac{l_{\max}+2\left\lfloor\frac{n}{2}\right\rfloor}{2}\right\}$ *with* $S[m_0] = 0$ *define* $r'_{\min} = 2m_0 - \min\left(l_{\max}, 2\left\lfloor\frac{m_0}{4}\right\rfloor, 2\left(\left\lceil\frac{3m_0-n}{4}\right\rceil - 1\right)\right)$ *and* $r'_{\max} = 2m_0 - \max\left(l_{\min}, 2\left(m_0 - \left\lfloor\frac{n}{2}\right\rfloor\right)\right)$. *There is an even* 3*-cadence which uses* m_0 *as middle element if and only if* $S\left[\{r'_{\min}, r'_{\min}+2, \dots, r'_{\max}\}\right]$ *contains a* 0.

Similarly to the case of the L-R-3-cadence, we can use this lemma to shrink the interval in which the last element of the arithmetic progression can be.

Corollary 3. *Let* S *be a binary string. Define* $R = \{r_{\min}, r_{\min}+1, \dots, n\}$ *for an* $r_{\min} \geq \left\lfloor\frac{2}{3}n\right\rfloor + 1$. *Let further* S_{even} *be of the form* $0^i S'$ *and let* $l_{\min} = 2$ *and* $l_{\max} = \max\left(2i, 2\left\lfloor\frac{1}{6}n\right\rfloor\right)$ *be the first and the last indices of the first run of* S_{even} *which can be the first index of a* 3*-cadence, respectively. Define the interval* $L = \{l_{\min}, l_{\min}+1, \dots, l_{\max}\}$.

If $S[R_{\text{even}}]$ *is of the form* 1^j*, there is no even* 3*-cadence starting in* L.

Otherwise, define $r_0 = \min\{r \in R_{\text{even}} | S[r] = 0\}$ *and use Lemma* 4 *to check whether there is a* 3*-cadence starting in* L *and ending with* r_0.

If such a 3*-cadence does not exist, define* $m_{\min} = \frac{l'_{\max}+r_0}{2}+1$ *with the variable* $l'_{\max} = \min\left(l_{\max}, 2\left\lfloor\frac{r_0}{6}\right\rfloor, 2\left(\left\lceil\frac{3r_0-2n}{2}\right\rceil - 1\right)\right)$ *as defined in Lemma* 4 *and define* $m_{\max} = \frac{l_{\max}+n}{2}$. *If* $S\left[\{m_{\min}, m_{\min}+1, \dots, m_{\max}\}\right]$ *is of the form* 1^j*, then there is no even* 3*-cadence starting in* L.

Otherwise, define $m_0 = \min\{m \in \{m_{\min}, m_{\min}+1, \dots, m_{\max}\} | S[m] = 0\}$. *Then use Lemma* 4 *to check whether there is a* 3*-cadence starting in* L *with* m_0 *as second element.*

If such a 3*-cadence does not exist, define* $R' = R \cap \mathbb{Z}_{>r'_{\max}}$ *with the variable* $r'_{\max} = 2m_0 - \max\left(l_{\min}, 2\left(m_0 - \left\lfloor\frac{n}{2}\right\rfloor\right)\right)$ *as defined in Lemma* 4. *There is an even* 3*-cadence starting in* L *if and only if there is an even* 3*-cadence starting in* L *and ending in* R'.

An application of this corollary can be seen in Fig. 3.

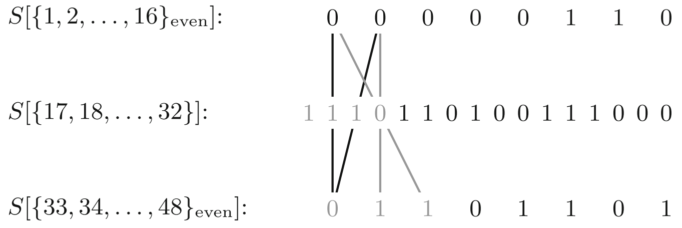


Fig. 3. A string with 48 characters after one application of Corollary 3. First, the index $r_0 = 34$ is found. The minimal and maximal candidates for 3-cadences with r_0 are given by the black lines. Then, the index $m_0 = 20$ is found. The minimal and maximal candidates for 3-cadences with m_0 are given by the gray lines. Afterwards, the gray characters are guaranteed not to form a 3-cadence with characters from the first run of S_{even}.

In the uncompressed case, each element of the middle third and the last third has to be read at most once in order to decide whether there is a 3-cadence which starts in the first run of S_{even}. Furthermore, we can modify this algorithm to detect the existence of a 3-cadence which start in the second run of S_{even}. By symmetry, we can also decide in linear time, whether there exists a 3-cadence which ends in one of the two last runs of S_{even}. Similarly, we can decide in linear time, whether there is an odd 3-cadence.

In the compressed case, a problem can arise if the first run of S_{even} is short. Let S_{even} be of the form $0^i 1 S'$. Then the 1 has index $2i+2$. Let r_1 be the smallest even index such that $(2i+2) - \frac{r_1-(2i+2)}{2} \leq 0$ and $(2i+2) + 3\frac{r_1-(2i+2)}{2} > n$ hold. Since $S[\{2,4,6,\ldots,2i+2\}]$ contains 01, we can use Corollary 1 and Lemma 2 to check in polynomial time, whether there is a 3-sub-cadence starting in $\{2,4,6,\ldots,2i+2\}$ and ending with an index greater than or equal to r_1. By construction, such a 3-sub-cadence would be a 3-cadence.

If such a 3-cadence exists, we are done. Therefore, it is only left to show that even in the compressed case, the application of Corollary 3 is fast enough to find a 3-cadence which start in the first run of S_{even} and end at an index smaller than r_1 in polynomial time if such a cadence exists. Since each application of Corollary 3 can be done in polynomial time, it is left to show that after a polynomial number of applications, the value $r'_{\max}$ is greater than r_1.

In the worst case, each r_0 is $r_{\min}$. Since each 3-sub-cadences with distance of at least $\frac{1}{3}n$ is a 3-cadence, we can assume $r_0 < r_1 \leq 2i + 2 + \frac{2}{3}n$ holds and therefore $l_{\max} \geq r_0 - \frac{2}{3}n$ holds as well. Also, both $2\left\lfloor\frac{r_0}{6}\right\rfloor$ and $2(\left\lceil\frac{3r_0-2n}{2}\right\rceil - 1)$ are greater than or equal to $r_0 - \frac{2}{3}n$. Therefore $l'_{\max} \geq r_0 - \frac{2}{3}n$ holds.

Similarly, in the worst case, each m_0 is directly behind $\frac{l'_{\max}+r_0}{2} \geq r_0 - \frac{1}{3}n$. Hence, we can assume $m_0 = 2r_0 - \frac{1}{3}n + 1$. With $l''_{\min} = \max\left(l_{\min}, 2\left(m_0 - \left\lfloor\frac{n}{2}\right\rfloor\right)\right)$, this implies that the inequality $2m_0 - l''_{\min} \geq \min\left(r_0 + (r_0 - \frac{2}{3}n), n-1\right)$ holds.

Therefore, for $r_0 < r_1$, one application of Corollary 3 checks for an interval of size $r_0 - \frac{2}{3}n$, whether there is 3-cadence which starts in the first run and ends in this interval. Therefore, we only need at most $\log n$ applications of this corollary.

This implies that it can be decided in polynomial time whether a compressed binary string contains any 3-cadences.

6 Conclusion

This paper shows that we can decide in linear time whether an uncompressed binary string contains a 3-cadence. While we should expect that it is more difficult to avoid 3-cadences in binary strings than to include 3-cadences, it is surprising that it is strictly easier to decide whether there is any 3-cadence at all than to decide whether there is a 3-cadence with a given character.

For the compressed case, we have shown that we can decide in polynomial time whether a compressed binary string contains a 3-cadence. However, all even slightly harder problems have been shown to be $\mathcal{NP}$-complete. These hardness-results seem to indicate that cadences may not be very useful in compressed pattern matching.

Regarding k-sub-cadences, there are no known nontrivial bounds on the bit complexity of the detection of k-sub-cadences with a given character. Closely related, it is unknown whether equidistant subsequence matching is $\mathcal{NP}$-hard on compressed binary strings.

Finally, in terms of uncompressed cadence detection, it is still unknown whether we can decide with sub-quadratic bit complexity whether a given string contains a 4-cadence. The currently best result is by Funakoshi et al., who presented in [3] a detection algorithm with sub-quadratic time complexity in the word RAM model.

References

1. Amir, A., Apostolico, A., Gagie, T., Landau, G.M.: String cadences. Theor. Comput. Sci. **698**, 4–8 (2017). https://doi.org/10.1016/j.tcs.2017.04.019
2. Charikar, M., et al.: The smallest grammar problem. IEEE Trans. Inf. Theor. **51**(7), 2554–2576 (2005). https://doi.org/10.1109/TIT.2005.850116
3. Funakoshi, M., Nakashima, Y., Inenaga, S., Bannai, H., Takeda, M., Shinohara, A.: Detecting k-(Sub-)cadences and equidistant subsequence occurrences. In: Gørtz, I.L., Weimann, O. (eds.) 31st Annual Symposium on Combinatorial Pattern Matching (CPM 2020). Leibniz International Proceedings in Informatics (LIPIcs), vol. 161, pp. 12:1–12:11. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, Dagstuhl, Germany (2020). https://doi.org/10.4230/LIPIcs.CPM.2020.12
4. Funakoshi, M., Pape-Lange, J.: Non-rectangular convolutions and (Sub-)cadences with three elements. In: Paul, C., Bläser, M. (eds.) 37th International Symposium on Theoretical Aspects of Computer Science (STACS 2020). Leibniz International Proceedings in Informatics (LIPIcs), vol. 154, pp. 30:1–30:16. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany (2020). https://doi.org/10.4230/LIPIcs.STACS.2020.30
5. Gardelle, J.: Cadences. Mathématiques et Sci. Humaines **9**, 31–38 (1964). http://www.numdam.org/item/MSH_1964__9__31_0
6. Jeż, A.: Faster fully compressed pattern matching by recompression. ACM Transactions on Algorithms 11(3), Jan 2015. https://doi.org/10.1145/2631920
7. Kempa, D., Kociumaka, T.: Resolution of the burrows-wheeler transform conjecture. CoRR abs/1910.10631. Accepted to the 61st Annual Symposium Foundations of Computer Science (FOCS 2020) (2019). http://arxiv.org/abs/1910.10631
8. Lohrey, M.: Algorithms on compressed words. The Compressed Word Problem for Groups. SM, pp. 43–65. Springer, New York (2014). https://doi.org/10.1007/978-1-4939-0748-9_3
9. Lothaire, M.: Combinatorics on Words. Cambridge Mathematical Library, Cambridge University Press, Cambridge (1997). https://doi.org/10.1017/CBO9780511566097
10. Pape-Lange, J.: On extensions of maximal repeats in compressed strings. In: Gørtz, I.L., Weimann, O. (eds.) 31st Annual Symposium on Combinatorial Pattern Matching, CPM 2020, June 17–19, 2020, Copenhagen, Denmark. LIPIcs, vol. 161, pp. 27:1–27:13. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2020). https://doi.org/10.4230/LIPIcs.CPM.2020.27
11. Rytter, W.: Application of Lempel-Ziv factorization to the approximation of grammar-based compression. Theor. Comput. Sci. **302**(1–3), 211–222 (2003). https://doi.org/10.1016/S0304-3975(02)00777-6
12. Beweis einer Baudet'schen Vermutung: Waerden, B.L.v.d. Nieuw Archief voor Wiskunde **15**, 212–216 (1927)