Subsequence Covers of Words

Panagiotis Charalampopoulos[1 [0000−0002−6024−1557]], Solon P. Pissis[2,3 [0000−0002−1445−1932]], Jakub Radoszewski[4,⋆ [0000−0002−0067−6401]], Wojciech Rytter[4 [0000−0002−9162−6724]], Tomasz Waleń[4,⋆⋆ [0000−0002−7369−3309]], and Wiktor Zuba[2 [0000−0002−1988−3507]]

[1] Birkbeck, University of London, London, UK p.charalampopoulos@bbk.ac.uk
[2] CWI, Amsterdam, The Netherlands {solon.pissis,wiktor.zuba}@cwi.nl
[3] Vrije Universiteit, Amsterdam, The Netherlands
[4] University of Warsaw, Warsaw, Poland {jrad,rytter,walen}@mimuw.edu.pl

Abstract. We introduce subsequence covers (s-covers, in short), a new type of covers of a word. A word C is an *s-cover* of a word S if the occurrences of C in S as subsequences cover all the positions in S.
The s-covers seem to be computationally much harder than standard covers of words (cf. Apostolico et al., *Inf. Process. Lett.* 1991), but, on the other hand, much easier than the related shuffle powers (Warmuth and Haussler, *J. Comput. Syst. Sci.* 1984).
We give a linear-time algorithm for testing if a candidate word C is an s-cover of a word S over a polynomially-bounded integer alphabet. We also give an algorithm for finding a shortest s-cover of a word S, which in the case of a constant-sized alphabet, also runs in linear time. Furthermore, we complement our algorithmic results with a lower and an upper bound on the length of a longest word without non-trivial s-covers, which are both exponential in the size of the alphabet.

Keywords: String algorithms · Combinatorics on words · Covers · Shuffle powers · Subsequence covers

1 Introduction

The problem of computing covers in a word is a classic one in string algorithms; see [1,2,11] and also [5] for a recent survey. In its most basic type, we say that a word C is a *cover* of another longer word S if every position of S lies within some occurrence of C as a factor (subword) in S [1].

In this paper we introduce a new type of cover, in which instead of subwords we take subsequences (scattered subwords). Such covers turn out to be related to shuffle problems [13,12,4]. Formally the new type of cover is defined as follows:

Definition 1. *A word C is a* **subsequence cover** *(s-cover, in short) of a word S if every position in S belongs to an occurrence of C as a subsequence in S. We also write $S \in C^{\otimes}$, where $C^{\otimes}$ is the set of words having C as an s-cover.*

⋆ Supported by the Polish National Science Center, grant no. 2018/31/D/ST6/03991.
⋆⋆ Supported by the Polish National Science Center, grant no. 2018/31/D/ST6/03991.

We say that an s-cover C of a word S is *non-trivial* if $|C| < |S|$. A word S is called *s-primitive* if it has no non-trivial s-cover.

An example s-primitive word is the Zimin word S_k [10], that is, a word over alphabet $\{1, \ldots, k\}$ given by recurrences of the form

$$S_1 = 1, \quad S_i = S_{i-1} i S_{i-1} \text{ for } i > 1.$$

The word S_k has length $2^k - 1$.

Clearly, if a word C is a (standard) cover of a word S, then C is an s-cover of S. However the converse implication is false: ab is an s-cover of aab, but is not a standard cover. For another example of an s-cover, see the following example.

Example 1. Fig. 1 shows that $C = abcab$ is an s-cover of $S = abcbacab$. In fact C is a shortest s-cover of S.

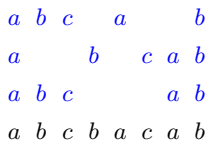

Fig. 1: An illustration of the fact that $C = abcab$ is an s-cover of $S = abcbacab$.

We now provide some basic definitions and notation. An *alphabet* is a finite nonempty set of elements called *letters*. A *word* S is a sequence of letters over some alphabet. For a word S, by $|S|$ we denote its *length*, by $S[i]$, for $i = 0, \ldots, |S| - 1$, we denote its ith letter, and by $Alph(S)$ we denote the set of letters in S, i.e., $\{S[0], \ldots, S[|S| - 1]\}$. The *empty word* is the word of length 0.

For any two words U and V, by $U \cdot V = UV$ we denote their concatenation. For a word $S = PUQ$, where P, U, and Q are words, U is called a *factor* of S; it is called a *prefix* (resp. *suffix*) if P (resp. Q) is the empty word. By $S[i \,..\, j]$ we denote a factor $S[i] \ldots S[j]$ of S; we omit i if $i = 0$ and j if $j = |S| - 1$.

A word V is a *k-power* of a word U, for integer $k \geq 0$, if V is a concatenation of k copies of U, in which case we denote it by U^k. It is called a *square* if $k = 2$.

Remark 1. If a word S contains a non-empty square factor U^2, then S has a non-trivial s-cover resulting by removing any of the two consecutive copies of U. Further, if a word S has a factor being a gapped repeat UVU (see [9]), such that $Alph(V) \subseteq Alph(U)$, then S has a non-trivial s-cover resulting by removing VU from the gapped repeat. Moreover, if C is an s-cover of S, then C is an s-cover of S concatenated with any concatenation of suffixes of C.

A different version of covers, where we require that position-subsequences are disjoint, is the *shuffle closure* problem. The shuffle closure of a word U, denoted

by $U^{\odot}$, is the set of words resulting by interleaving many copies of U; see [13]. The words in $U^{\odot}$ are sometimes called *shuffle powers* of U.

The following problems are NP-hard for constant-sized alphabets:

(1) Given two words U and S, test if $S \in U^{\odot}$; see [13].
(2) Given a word S, check if there exists a word U such that $|U| = |S|/2$ and $S \in U^{\odot}$ (this was originally called the *shuffle square* problem); see [4]. An NP-hardness proof for a binary alphabet was recently given in [3].
(3) Given a word S, find a shortest word U such that $S \in U^{\odot}$; its hardness is trivially reduced from (2).

The following observation links s-covers and shuffle closures.

Observation 1 *Let S be a word of length n. Then*

$$S \in C^{\otimes} \Leftrightarrow \exists\, r_0, r_1 \ldots, r_{n-1} \in \mathbb{Z}_+ :\ S[0]^{r_0} S[1]^{r_1} \ldots S[n-1]^{r_{n-1}} \in C^{\odot}.$$

In this paper we show that problems similar to (1) and (3) for s-covers, when we replace $\odot$ by $\otimes$, are tractable: notably, the first one is solved in linear time for any polynomially-bounded integer alphabet; and the last one in linear time for any constant-sized alphabet.

Our results and paper organization:

- In Section 2 we present a linear-time algorithm for checking if a word C is an s-cover of a word S, assuming that C and S are over a polynomially-bounded integer alphabet $\{0, \ldots, |S|^{\mathcal{O}(1)}\}$. We also discuss why an equally efficient algorithm for this problem without this assumption is unlikely.

- Let $\gamma(k)$ denote the length of a longest s-primitive word over an alphabet of size k. In Section 3 we present general bounds on this function as well as its particular values for small values of k.

- In Section 4 we show that computing a non-trivial s-cover is fixed-parameter tractable for parameter $k = |Alph(S)|$. In particular we obtain a linear-time algorithm for computing a shortest s-cover of a word over a constant-sized alphabet.

- Finally in Section 5 we explore properties of s-covers that are significantly different from properties of standard covers. In particular, we show that a word can have exponentially many different shortest s-covers, which implies that computing all shortest s-covers of a word (over a superconstant alphabet) requires exponential time.

2 Testing if a word is an s-cover

Consider words $C = C[0\,..\,m-1]$ and $S = S[0\,..\,n-1]$. We would like to check whether C is an s-cover of S.

Let sequences $\mathit{FirstOcc} = (p_1, p_2, \ldots, p_m)$ and $\mathit{LastOcc} = (q_1, q_2, \ldots, q_m)$ be the lexicographically first and last position-subsequences of S containing C, where $p_1 = 0$ and $q_m = n-1$. If there are no such subsequences of positions then C is not an s-cover, so we assume they exist and are well defined.

For all $i \in \{0, \ldots, n-1\}$, we define

$$\mathit{Right}[i] = \min(\{j \,:\, q_j > i\} \cup \{m+1\}),$$
$$\mathit{Pref}[i] = \max(\{j \,:\, p_j \leq i \ \wedge\ S[p_j] = S[i]\} \cup \{0\}).$$

Intuitively, if position i is in any subsequence occurrence of C in S, then there is a subsequence occurrence of C in S that consists of the prefix of $\mathit{FirstOcc}$ of length $\mathit{Pref}[i]$ and an appropriate suffix of $\mathit{LastOcc}$. All we have to do is check, for all i, whether such a pair of prefix and suffix exists. See Fig. 2 for an illustration of the argument and Lemma 1 for a formal statement of the condition that needs to be satisfied.

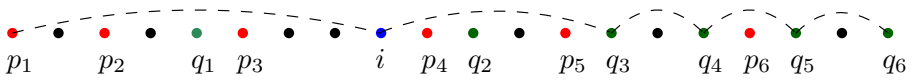

Fig. 2: Assume that for some words C and S the sequences $\mathit{FirstOcc}$ (red) and $\mathit{LastOcc}$ (green) are as in the figure. Further assume that $\mathit{Pref}[i] = 2$ (i.e., we have $S[i] = S[p_2] \neq S[p_3]$). As shown, we have $\mathit{Right}[i] = 2$. Thus, we have $\mathit{Right}[i] \leq \mathit{Pref}[i]+1$ and consequently the position i is covered by an occurrence of C as a subsequence using positions $(p_1, i, q_3, q_4, q_5, q_6)$.

Lemma 1. *Let us assume that FirstOcc and LastOcc are well defined. Then C is an s-cover of S if and only if for each position $0 \leq i \leq n-1$ we have: $\mathit{Pref}[i] > 0$ and $\mathit{Right}[i] \leq \mathit{Pref}[i] + 1$.*

Proof. First, observe that if $\mathit{Pref}[i] = 0$ for any i, then C is not an s-cover of S. This follows from the greedy computation of $\mathit{FirstOcc}$, which implies that the prefix of C that precedes the first occurrence of $S[i]$ in C does not have a subsequence occurrence in $S[0\,..\,i-1]$; else, i would be in $\mathit{FirstOcc}$, a contradiction.

We henceforth assume that $\mathit{Pref}[i] > 0$ for every i and show that, in this case, C is an s-cover of S if and only if $\mathit{Right}[i] \leq \mathit{Pref}[i]+1$ for all $i \in \{0, \ldots, n-1\}$.

($\Leftarrow$) Assume that $\mathit{Right}[i] \leq \mathit{Pref}[i]+1$. In this case position i can be covered by a subsequence occupying positions $p_1, \ldots, p_{j-1}, i, q_{j+1}, \ldots, q_m$, for $j = \mathit{Pref}[i]$. As $S[p_j] = S[i]$ this subsequence is equal to C, and as $p_j \leq i$ and $q_{j+1} > i$ ($j+1 \geq \mathit{Right}[i]$) those positions form an increasing sequence (that is, we obtain a valid subsequence).

($\Rightarrow$) On the other hand assume that for some j there exists an increasing sequence

$$r_1, r_2, \ldots, r_{j-1}, i, r_{j+1}, \ldots, r_m,$$

such that $S[r_1]S[r_2]\ldots S[r_{j-1}]S[i]S[r_{j+1}]\ldots S[r_m] = C$.

By induction for $k = 1, \ldots, j$, $r_k \geq p_k$ (including $r_j = i$) and for $k = m, \ldots, j+1$, $r_k \leq q_k$. But this means that $\mathit{Pref}[i] \geq j$ and $\mathit{Right}[i] \leq j+1$. Hence $\mathit{Right}[i] \leq \mathit{Pref}[i] + 1$. This completes the proof. □

The sequence *FirstOcc* can be computed with a simple left-to-right pass over S and C; the computation of *LastOcc* is symmetric. The table *Right* can be computed via a right-to-left pass. The table $\mathit{Pref}[i]$ is computed on-line using an additional table *PRED* indexed by the letters of the alphabet. The algorithm is formalized in the following pseudocode.

Algorithm 1: $\mathit{TEST}(C, S)$

Input: word $C = C[0\,..\,m-1]$ and word $S = S[0\,..\,n-1]$
Output: *true* if and only if C is an s-cover of S

compute $\mathit{FirstOcc} = (p_1, \ldots, p_m)$ and $\mathit{LastOcc} = (q_1, \ldots, q_m)$

▷ compute *Right*
- $k := m + 1$
- **for** $i := n - 1$ **down to** 0 **do**
  - $\mathit{Right}[i] := k$
  - **if** $k > 1$ **and** $i = q_{k-1}$ **then** $k := k - 1$

▷ compute *Pref*
- $\mathit{PRED}[c] := 0\ \forall c \in \Sigma$
- $k := 1$
- **for** $i := 0$ **to** $n - 1$ **do**
  - **if** $i = p_k$ **then**
    - $\mathit{PRED}[S[i]] := k$
    - **if** $k < m$ **then** $k := k + 1$
  - $\mathit{Pref}[i] := \mathit{PRED}[S[i]]$

return $\forall_{i=0,\ldots,n-1}\,(\,\mathit{Pref}[i] > 0$ **and** $\mathit{Right}[i] \leq \mathit{Pref}[i] + 1\,)$

The correctness of the algorithm follows from Lemma 1 (inspect also Fig. 2). Note that, under the assumption of a polynomially-bounded integer alphabet, the table *PRED* can be initialized and updated deterministically in linear total time by first sorting the letters of S. We thus arrive at the following result.

Theorem 1. *Given words C and S over an integer alphabet $\{0, \ldots, |S|^{\mathcal{O}(1)}\}$, we can check if C is an s-cover of S in $\mathcal{O}(|S|)$ time.*

In the standard setting (cf. [2]), one can check if a word C is a cover of a word S—what is more, find the shortest cover of S—in linear time for any (non-necessarily integer) alphabet. We show below that the existence of such an algorithm for testing a candidate s-cover is rather unlikely.

Let us introduce a slightly more general version of the s-cover testing problem in which, if C is an s-cover of S, we are to say, for each position i in S, which position j of C is actually used to cover $S[i]$; if there is more than one such position j, any one of them can be output. Let us call this problem the *witness s-cover testing* problem. In particular, our algorithm solves the witness s-cover testing problem with the answers stored in the *Pref* array. Actually it is hard to imagine an algorithm that solves the s-cover testing problem and not the witness version of it. We next give a comparison-based lower bound for the latter.

Theorem 2. *The witness s-cover testing problem for a word S of length n requires $\Omega(n \log n)$ time in the comparison model.*

Proof. Let us consider a word C of length m that is composed of m distinct letters and a family of words of the form $S = CTC$, where T is a word of length m such that $Alph(T) \subseteq Alph(C)$. Then C is an s-cover of each such word S. Each choice of word T implies a different output to the witness s-cover testing problem on C and S. There are m^m different outputs, so a decision tree for this problem must have depth $\Omega(\log m^m) = \Omega(m \log m) = \Omega(n \log n)$. □

Let us further notice that even if C turns out not to be an s-cover of S, our algorithm actually computes the positions of S that can be covered using occurrences of C (they are exactly the positions i for which $Pref[i] > 0$ and $Right[i] \leq Pref[i]+1$). Hence our algorithm may be useful to find partial variants of s-covers, defined analogously as for the standard covers [6,7,8].

3 Maximal lengths of s-primitive words

Let us recall that $\gamma(k)$ denotes the length of a longest s-primitive word over an alphabet of size k. It is obvious that $\gamma(2) = 3$; the longest s-primitive binary words are *aba* and *bab*. The case of ternary words is already more complicated; we study it in Section 3.1. General bounds on the function $\gamma(k)$ are shown in Section 3.2. A discussion on computing $\gamma(k)$ for small $k > 3$ is presented in Section 3.3. In particular, we were not able to compute the exact value of $\gamma(5)$.

3.1 Ternary alphabet

Fact 1 $\gamma(3) = 8$.

Proof. The word $S = abcabacb$ is of length 8 and it is s-primitive, hence $\gamma(3) \geq 8$.

We still have to show that each 3-ary word of length 9 is not s-primitive (there are 19683 ternary words). The number of words to consider is substantially reduced by observing that relevant words are square-free and do not contain the structure specified in the following claim.

Claim. If a word S over a ternary alphabet contains a factor of the form $abXbc$ for some (maybe empty) word X and different letters a, b, c, then it is not s-primitive.

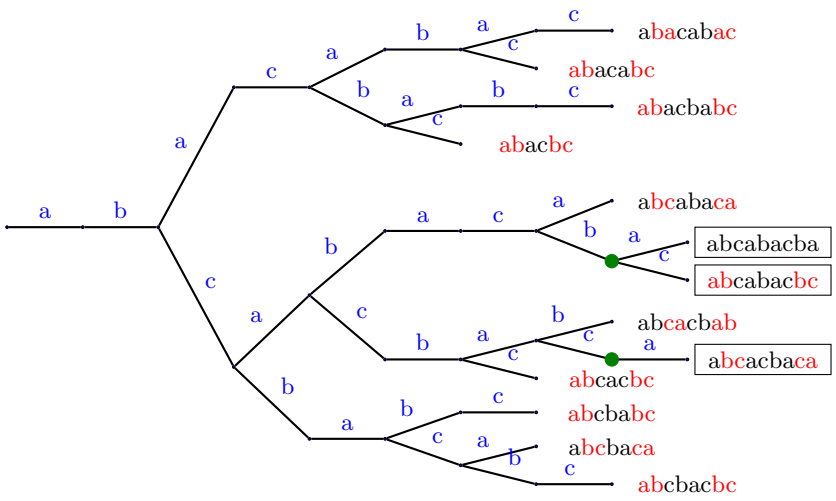

Fig. 3: A trie of all ternary square-free words starting with ab, truncated at words that are not s-primitive (in leaves). Only one word in a leaf ($abcabacba$) does not contain the structure specified in the claim inside the proof of Fact 1, but it still has a non-trivial s-cover $abcba$. The trie has depth 9 (the leaves with words of length 9 are shown in frames and the internal nodes of depth 8 corresponding to s-primitive words are drawn as green circles), so $\gamma(3) = 8$.

Proof. The factor $abXbc$ has abc as its s-cover, and thus it is not s-primitive. Consequently, the whole word S is not s-primitive. □

Fig. 3 shows a trie of all ternary square-free words starting with ab, truncated at words that are not s-primitive (in leaves). The words in all leaves but one contain the structure from the claim, and for the remaining word, a non-trivial s-cover can be easily given. The trie shows that words of length 9 over a ternary alphabet are not s-primitive. □

3.2 General alphabet

Definition 2. *For a word S over alphabet $Alph(S)$ of size k, let $first(S)$ (resp. $last(S)$) denote the length-k word containing all the letters of $Alph(S)$ in the order of their first (resp. last) occurrence in S.*

Example 2. $first(\underline{ab}a\underline{d}\underline{bc}d) = abdc$, $last(ab\underline{a}d\underline{bcd}) = abcd$.

Lemma 2. *Let C be an s-cover of S. Then $first(C) = first(S)$ and $last(C) = last(S)$.*

Proof. Assume that letter a appears before letter b in $first(S)$, but after letter b in $first(C)$. Then $Pref[i] = 0$ (see Section 2 for the definition) for $i = \min\{j : S[j] = a\}$. This proves that $first(C) = first(S)$; a proof that $last(C) = last(S)$ follows by symmetry. □

Example 3. Using a computer one can check that $S = abacadbabdcabcbadac$ is an s-primitive word of length 19 over a quaternary alphabet. Thus $\gamma(4) \geq 19$.

For a word X we define X_- (resp. X^-) as the word obtained from X by deleting the first (resp. last) letter.
By $shrink(S)$ we denote the word obtained from S by merging any non-zero number of consecutive copies of the same letter into just one copy. For example $shrink(abbacccbdd) = abacbd$. We define

$$\mathsf{FaLaFeL}(S) = shrink(F \cdot L^- \cdot F_- \cdot L), \text{ where } F = first(S),\ L = last(S).$$

Example 4. For $S = ababbacbcaabb$ we have

$$F = first(S) = abc,\ L = last(S) = cab,\ F_- = bc,\ L^- = ca, \text{ and}$$

$$shrink(F\,L) = abcab, \quad \mathsf{FaLaFeL}(S) = shrink(abc\,ca\,bc\,cab) = abc\,a\,bc\,ab.$$

Observation 2 *The word $shrink(F\,L)$ is an s-cover of $\mathsf{FaLaFeL}(S)$. However, it is possible that $shrink(F\,L)$ is an s-cover of S, while $\mathsf{FaLaFeL}(S)$ is not (as in the example).*

Lemma 3. *If the word $\mathsf{FaLaFeL}(S)$ is a subsequence of S, then $shrink(FL)$ is an s-cover of S.*

Proof. We need to show that each position i of S is covered by an occurrence of $shrink(FL)$ as a subsequence.

There exists a position j in S such that $shrink(FL^-)$ is a subsequence of $S[..j]$ and $shrink(F_-L)$ is a subsequence of $S[j..]$. We can assume that $i \leq j$; the other case is symmetric.

Let p be the index such that $F[p] = S[i]$. It suffices to argue that:

(1) $F[..p-1]$ is a subsequence of $S[..i-1]$; and

(2) $shrink(FL)[p+1..]$ is a subsequence of $S[i+1..]$.

Point (1) follows by the definition of $F = first(S)$.

As for point (2), if $i < j$, then $S[i+1..]$ has a subsequence $shrink(F_-L)$ by the definition of j and $shrink(FL)[p+1..]$ is a suffix of $shrink(F_-L)$.

If $p > 0$, then $S[i..]$ has a subsequence $shrink(F_-L)$ and so $S[i+1..]$ has a subsequence $shrink(FL)[2..]$.

Finally if $i = j$ and $p = 0$, then $S[i+1..]$ has a subsequence $shrink(F_-L)$ because $F_-[0] \neq F[0] = S[i]$. □

We will apply the following lemma for $Z = \mathsf{FaLaFeL}(S)$.

Lemma 4. *Let S, Z be words and x be a positive integer such that $|Alph(S)| = k$, $|S| = 2kx + 1$ and $|Z| \leq 4k - 2$. We assume that each factor of S of length $x + 1$ contains all k letters, and the length-$(x + 1)$ prefix/suffix of S contains, as a subsequence, the length-k prefix/suffix of Z, respectively. If $shrink(Z) = Z$, then S contains Z as a subsequence.*

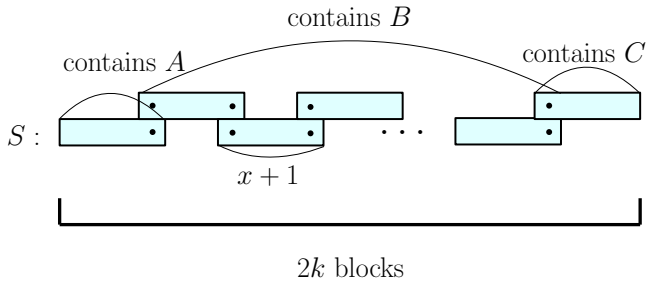

Fig. 4: Illustration of the proof of Lemma 4. Let $Z = ABC$ where $|A| = |C| = k$. Each block represents a factor of length $x+1$ containing all letters and starting at a given position. The blocks overlap by one letter. We have $|S| = 2kx + 1$.

Proof. Let us cover S with $2k$ blocks, each of length $x+1$, with overlaps of one position between consecutive blocks; see Fig. 4.

Let $Z = ABC$ where $|A| = |C| = k$. By the assumption of the lemma, the first and the last block in S contain A and C as a subsequence, respectively. Let us choose some $2k$ positions in S that form these occurrences. Each of the remaining $2k-2$ blocks contains a copy of each of the letters in $Alph(S)$; in particular, we can choose the letters from the word B in them. No two consecutive letters in B are the same, so we will not choose the same position twice. □

Observation 3 *If a letter a occurs in a word $S = S'aS''$ only once, then every s-cover of S has a form $C'aC''$, where C', C'' are s-covers of S', S'', respectively.*

Theorem 3. *For $k \geq 4$ we have*

$$5 \cdot 2^{k-2} - 1 \leq \gamma(k) \leq 2^{k-1}\, k!.$$

Proof. We separately prove the lower and upper bounds.

Lower bound. We can take the sequence of words $S_4 = abacadbabdcabcbadac$, and for $k > 4$:

$$S_k = S_{k-1} a_k S_{k-1}, \text{ where } a_k \text{ is a new letter.}$$

We have $|S_k| = 5 \cdot 2^{k-2} - 1$, and S_k has no non-trivial s-cover, due to Observation 3 and Example 3. Hence $\gamma(k) \geq 5 \cdot 2^{k-2} - 1$.

Upper bound. We will show that

$$\gamma(k) \leq 2k \cdot \gamma(k-1). \tag{1}$$

Let us assume that $|Alph(S)| = k$ and $|S| > 2k \cdot \gamma(k-1) + 1$. Let $x = \gamma(k-1)$. If any factor U of S of length $x+1$ does not contain all k letters, then U is not s-primitive by the definition of γ. If $S = PUQ$ where U has a non-trivial s-cover C, then PCQ is a non-trivial s-cover of S and, consequently, S is not s-primitive.

Otherwise, by Lemma 4 applied for a prefix of S of length $2kx + 1$ and $Z = \mathsf{FaLaFeL}(S)$, $\mathsf{FaLaFeL}(S)$ is a subsequence of S. By Lemma 3, $shrink(FL)$ is an s-cover of S. It is non-trivial as for $k \geq 3$, $shrink(FL)$ is shorter than $\mathsf{FaLaFeL}(S)$. In either case, S is not s-primitive and (1) holds. Using a simple induction we get $\gamma(k) \leq 2^{k-1}k!$. □

3.3 Behaviour of the function $\gamma(k)$ for small k

The values of γ for small k are as follows (see also Table 1):

- $\gamma(1) = 1$ – trivial;
- $\gamma(2) = 3$ – using square-free words;
- $\gamma(3) = 8$ – due to Fact 1 and Fig. 3;
- $\gamma(4) = 19$ – through computer experiments[5];
- $39 \leq \gamma(5) \leq 190$ – due to Inequality (1) and $\gamma(4) = 19$.

k	$\gamma(k)$	examples of s-primitive words
1	1	*a*
2	3	*aba*
3	8	*abcabacb*
4	19	*abacadbabdcabcbadac* *abcdabacadbdcbabdac*
5	≥ 39	*abacadbabdcabcbadaceabacadbabdcabcbadac*

Table 1: The values of γ for small alphabet-size k.

Remark 2. There are $2 \cdot 3! = 12$ s-primitive words of length $\gamma(3) = 8$ over ternary alphabet (cf. Section 3.1 and Fig. 3, for each pair of distinct letters there are two s-primitive words starting with these letters). This accounts for less than 0.2% among all 3^8 ternary words of length 8. For a 4-letter alphabet, our program shows that the relative number of s-primitive words of length $\gamma(4) = 19$ is very small. There are exactly 2496 such words, out of 4^{19}, which gives a fraction less than 10^{-8}. This suggests that s-primitive 5-ary words of length $\gamma(5)$ are extremely sparse and finding an s-primitive word over a 5-letter alphabet of length $\gamma(5)$, if $\gamma(5) > 39$, could be a challenging task.

4 Computing s-covers

The following observation is a common property of s-covers and standard covers.

Observation 4 *If C is an s-cover of S and C' is an s-cover of C, then C' is an s-cover of S.*

Theorem 4. *Let S be a length-n word over an integer alphabet of size $k = n^{\mathcal{O}(1)}$.*

[5] The optimized `C++` code used for the experiments can be found at `https://www.mimuw.edu.pl/~jrad/code.cpp`. The program reads k and computes $\gamma(k)$; it finishes within 1 minute for $k \leq 4$.

(a) A shortest s-cover of S can be computed in $\mathcal{O}(n \cdot \min(2^n, k^{\gamma(k)}))$ time.
(b) One can check if S is s-primitive and, if not, return a non-trivial s-cover of S in $\mathcal{O}(n + 2^{\gamma(k)}\gamma(k))$ time.
(c) An s-cover of S of length at most $\gamma(k)$ can be computed in $\mathcal{O}(n2^{\gamma(k)}\gamma(k))$ time.

Proof. (a) By Theorem 3, there are $\mathcal{O}(k^{\gamma(k)})$ s-primitive k-ary words and, by Observation 4, the shortest s-cover of S must be one of them. On the other hand, there are 2^n subsequences of S. Hence, there are $\min(2^n, k^{\gamma(k)})$ candidates to be checked. With the aid of the algorithm from Theorem 1 we can check each candidate in $\mathcal{O}(n)$ time. This gives the desired complexity.

(b) If $n \leq \gamma(k)$, we can use the algorithm from (a) which works in $\mathcal{O}(2^{\gamma(k)}\gamma(k))$ time. Otherwise, we know by Theorem 3 that S is not s-primitive. We can find a non-trivial s-cover of S as follows. Let $S = S'S''$ where $|S'| = \gamma(k) + 1$. We can use the algorithm from (a) to compute a shortest s-cover C of S' in $\mathcal{O}(2^{\gamma(k)}\gamma(k))$ time. By Theorem 3, C is a non-trivial s-cover of S'. Then, we can output CS'' as a non-trivial s-cover of S. This takes $\mathcal{O}(n + 2^{\gamma(k)}\gamma(k))$ time.

(c) By Observation 4, any s-cover of an s-cover of S will be an s-cover of S. We can thus repeatedly apply the algorithm underlying (b); apart from outputting the computed non-trivial s-cover. As each application of this algorithm removes at least one letter of S, the number of steps is at most $n - \gamma(k)$. Each step takes $\mathcal{O}(2^{\gamma(k)}\gamma(k))$ time and hence the conclusion follows. □

Corollary 1. *A shortest s-cover of a word over a constant-sized alphabet can be computed in linear time.*

5 The number of distinct shortest s-covers

In the case of standard covers, if a word S has two covers C, C', then one of C, C' is a cover of the other. This property implies, in particular, that a word has exactly one shortest cover.

In this section we show that analogous properties do not hold for s-covers. There exist words S having two s-covers C, C' such that none of C, C' is an s-cover of the other; e.g. $S = abcabcabcb$, $C = abcb$ and $C' = abcacb$. Moreover, a word can have many different shortest s-covers, as shown in Theorem 5.

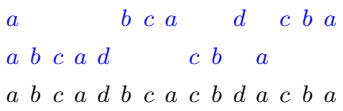

Fig. 5: $C_1 = abca\ d\ cba$ is a shortest s-cover of $S = abca\ d\ bcacb\ d\ acba$. S is a palindrome, hence $C_2 = abc\ d\ acba$ is also its s-cover.

Theorem 5. *For every positive integer n there exists a word of length n over an alphabet of size $\mathcal{O}(\log n)$ that has at least $2^{\lfloor \frac{n+1}{16} \rfloor}$ different shortest s-covers.*

Proof. We start with an example of a word with two different shortest s-covers and then extend it recursively.

Claim. The word $S = abca\ d\ bcacb\ d\ acba$ has two different s-covers of length 8, $C_1 = abca\ d\ cba$ and $C_2 = abc\ d\ acba$ (cf. Fig. 5). It does not have any shorter s-cover.

Proof. Any s-cover of this word must contain the letter d and before its first occurrence letters a, b, c (in that order) must appear. Symmetrically, after this letter, letters c, b, a must appear. The only word of length smaller than 8 which satisfies this property is $abc\ d\ cba$; however, this is not an s-cover of S (as it does not cover the middle letter a in S). □

We now construct a sequence of words T_i such that $T_0 = S$ and $T_i = T_{i-1}a_iT_{i-1}$ for $i > 0$, where a_i is a new letter.

The word T_i has length $16 \cdot 2^i - 1 = 2^{i+4} - 1$. Let us consider an infinite word $T = \lim_{i\to\infty} T_i$ (this word is well defined as each T_i is a prefix of T_{i+1}).

We show by induction, using Observation 3, that $T[0\,..\,n-1]$ has at least $2^{\lfloor \frac{n+1}{16} \rfloor}$ different shortest covers.

The base case for $n \leq 15$ holds as every word has a shortest s-cover and for $n = 15$ we apply the previous claim as $T[..\,n-1] = S$. Assume that $n > 15$. Let i be a non-negative integer such that $2^{i+4} \leq n < 2^{i+5}$. Then $T[0\,..\,n-1] = T_{i+1}[..\,n-1] = T_i\,a_i\,T_i[..\,n-2^{i+4}-1]$. By Observation 3, the number of shortest s-covers of $T[..\,n-1]$ is the number of shortest s-covers of T_i times the number of shortest s-covers of $T[..\,n-2^{i+4}-1]$, that is, at least

$$2^{\frac{2^{i+4}}{16}} \cdot 2^{\lfloor \frac{n-2^{i+4}+1}{16} \rfloor} = 2^{\lfloor \frac{n+1}{16} \rfloor}, \text{ as desired.}$$ □

6 Final remarks

There are several natural questions concerning the following problems:

1. Is a given word s-primitive?
2. What is its shortest s-cover?
3. What is the number of its different s-covers?
4. What is the exact value of $\gamma(5)$?
5. Let us define $\gamma'(1) = 1,\ \gamma'(k+1) = 2\,\gamma'(k) + k$ for $k > 1$.

   We have $\gamma(k) = \gamma'(k)$ for $1 \leq k < 5$. Is it always true?
6. Is there a really short, understandable and computer-avoiding proof of s-primitiveness of the word $a\,b\,a\,c\,a\,d\,b\,a\,b\,d\,c\,a\,b\,c\,b\,a\,d\,a\,c$?

We believe that the first three problems are NP-hard for general alphabets.

Acknowledgements

We thank Juliusz Straszyński for his help in conducting computer experiments.

This version of the contribution has been accepted for publication, after peer review but is not the Version of Record and does not reflect post-acceptance improvements, or any corrections. The Version of Record is available online at: https://doi.org/10.1007/978-3-031-20643-6_1. Use of this Accepted Version is subject to the publisher's Accepted Manuscript terms of use https://www.springernature.com/gp/open-research/policies/accepted-manuscript-terms.

References

1. 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
2. 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
3. Bulteau, L., Vialette, S.: Recognizing binary shuffle squares is NP-hard. Theor. Comput. Sci. **806**, 116–132 (2020). https://doi.org/10.1016/j.tcs.2019.01.012
4. Buss, S., Soltys, M.: Unshuffling a square is NP-hard. J. Comput. Syst. Sci. **80**(4), 766–776 (2014). https://doi.org/10.1016/j.jcss.2013.11.002
5. 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
6. Flouri, T., Iliopoulos, C.S., Kociumaka, T., Pissis, S.P., Puglisi, S.J., Smyth, W.F., Tyczynski, W.: Enhanced string covering. Theor. Comput. Sci. **506**, 102–114 (2013). https://doi.org/10.1016/j.tcs.2013.08.013
7. 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
8. Kociumaka, T., Pissis, S.P., Radoszewski, J., Rytter, W., Walen, 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
9. Kolpakov, R., Podolskiy, M., Posypkin, M., Khrapov, N.: Searching of gapped repeats and subrepetitions in a word. In: 25th Annual Symposium on Combinatorial Pattern Matching, CPM 2014. pp. 212–221 (2014). https://doi.org/10.1007/978-3-319-07566-2_22
10. Lothaire, M.: Algebraic Combinatorics on Words. Encyclopedia of Mathematics and its Applications, Cambridge University Press (2002). https://doi.org/10.1017/CBO9781107326019
11. 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
12. Rizzi, R., Vialette, S.: On recognizing words that are squares for the shuffle product. In: 8th International Computer Science Symposium in Russia, CSR 2013. Lecture Notes in Computer Science, vol. 7913, pp. 235–245. Springer (2013). https://doi.org/10.1007/978-3-642-38536-0_21
13. Warmuth, M.K., Haussler, D.: On the complexity of iterated shuffle. J. Comput. Syst. Sci. **28**(3), 345–358 (1984). https://doi.org/10.1016/0022-0000(84)90018-7