Algorithmic View on Circular String Attractors

Giuseppe Romana[1,*]

[1]University of Palermo, Italy

Abstract

The notion of circular string attractor has been recently introduced by Mantaci et al. [TCS 2021]. It consists of a set Γ_c of positions in a word such that each distinct circular factor has at least an occurrence crossing one of the elements of Γ_c. Its definition is an extension of the notion of string attractor by Kempa and Prezza [STOC 2018], which has been introduced as a unifying framework for some dictionary-based compressors.

In this paper, we present the first linear time algorithm to check whether a set is a circular string attractor of a word $w \in \{a_1, \ldots, a_\sigma\}^n$ by using $O(n \log n)$ bits of space. We further show that, for each $p > 0$, the decision problem of having a circular string attractor of size $\leq p$ is NP-complete. The proof is obtained through a reduction from the analogous problem for string attractors, for which Kempa and Prezza [STOC 2018] proved the NP-completeness. This reduction naturally leads to a new algorithm for checking whether a set is a string attractor that, unlike one of the solutions proposed by Kempa et al. [ESA 2018], it is independent from the size of the alphabet.

Keywords

Circular String Attractor, String Attractor, Suffix Array, Conjugate Array, Longest Common Prefix Array

1. Introduction

The notion of string attractor has been introduced by Kempa and Prezza [1] in the fields of data compression and indexed data structures. Given an alphabet $\Sigma = \{a_1, a_2, \ldots, a_\sigma\}$, an integer $n > 0$, and a word $w \in \Sigma^n$, the *string attractor* of a word is a subset Γ of positions in w such that every distinct factor of w has at least one occurrence crossing a position in Γ. The measure γ^*, which counts the size of the smallest string attractor of a word, has been subject of different combinatorial studies [2, 3, 4, 5, 6]. From an algorithmic perspective, given an integer $p > 0$, it has been proved that the decision problem of finding a string attractor of size $\leq p$ is NP-complete [1, 7].

Later, Mantaci et al. [2] introduced the notion of *circular string attractor*: instead of considering only the factors that occur within words, they further take into account the (circular) factors that appear on some conjugate of the word, and they denote by γ_c^* the corresponding size of a smallest circular string attractor. They further proved that for every word w, while it holds that the bound $\gamma_c^*(w) \in O(\gamma^*(w))$ is tight, the measure γ_c^* can be asymptotically smaller than γ^*.

In this work, we present the first algorithm for checking if a set Γ_c is a circular string attractor for a n-length word w. Such an algorithm operates in time $O(n)$ using $O(n \log n)$ bits. First we

ICTCS 2023: 24th Italian Conference on Theoretical Computer Science, September 13–15, 2023, Palermo, IT

*Corresponding author.

giuseppe.romana01@unipa.it (G. Romana)
0000-0002-3489-0684 (G. Romana)

show the properties that the data structures used verify, then we use these results to prove the correctness of the algorithm proposed. Later, we analyze the complexity of the decision problem of finding a circular string attractor of a word w of size at most $p > 0$. This complexity is obtained through a reduction from analogous problem on classical string attractor, leading to a new algorithm for checking whether a set Γ is a string attractor. Even if the bounds obtained are analogous to those of one of the algorithms presented by [7], we use well-known and simpler data structures widely used in combinatorial pattern matching, namely the Suffix Array and the Longest Common Prefix Array.

2. Preliminaries

An *alphabet* Σ is a set of elements called *letters* (or *characters*). We assume Σ to be finite, and we denote by $\sigma = |\Sigma|$ its cardinality. A *word* (or *string*) w is a sequence of letters from Σ, and its *length* is denoted by $|w|$. We denote with ε the *empty word*, that is the only word such that $|\varepsilon| = 0$. We denote by Σ^* the set of all words over Σ, by $\Sigma^+ = \Sigma^* \setminus \{\varepsilon\}$ the set of all non-empty words over Σ, and by $\Sigma^n = \{w \in \Sigma^* \mid |w| = n\}$ the set of all words of length n, for some integer $n > 0$. We denote by $\mathsf{alph}(w) \subseteq \Sigma$ the set of letters from Σ that occur in w.

For each pair $i, j \in [1, n]$, we denote by $w[i, j]$ the *factor* (or *substring*) of w starting at position i and ending at position j. Further, if $i = 1$ or $j = n$, then $w[i, j]$ is called *prefix* or *suffix*, respectively. Note that $w[1, n] = w$, and if $i > j$, we assume that $w[i, j] = \varepsilon$. The set of factors of a word w is denoted by $\mathcal{F}(w)$.

Given two words $u, v \in \Sigma^*$, if $u = u[1]u[2] \cdots u[|u|]$ and $v = v[1]v[2] \cdots v[|v|]$, the *concatenation* of u and v, denoted by $u \cdot v$ or simply uv, is the word $u[1]u[2] \cdots u[|u|] \cdot v[1]v[2] \cdots v[|v|]$.

If a total order on the elements of Σ is defined, then we can induce different orders on words. Given two words $u, v \in \Sigma^*$, we say that $u \leq v$ if u is prefix of v, or there exists a word w and two letters $a < b \in \Sigma$ such that wa and wb are prefixes of u and v respectively. We refer to this order of the words as *lexicographical order*.

Given two words $w_1, w_2 \in \Sigma^*$, we say that w_1 is a *conjugate* (or *cyclic rotation*, or simply *rotation*) of w_2 if there exist $u, v \in \Sigma^*$ such that $w_1 = uv$ and $w_2 = vu$. We denote by $\mathcal{R}(w)$ the set of rotations of the word w. Note that $\mathcal{R}(w_1) = \mathcal{R}(w_2)$ if and only if w_1 and w_2 are respectively conjugate. A finite word w is then *primitive* if and only if $|\mathcal{R}(w)| = |w|$, i.e., when all rotations of w are distinct. A *circular factor* of a word w is a factor that occur in at least one of the rotations of w. The set of all circular factors of a word w is denoted by $\mathcal{C}(w) = \bigcup_{w' \in \mathcal{R}(w)} \mathcal{F}(w')$.

The *Suffix Array* (SA) of a word $w \in \Sigma^n$ is an array of length n such that $w[\mathsf{SA}[i], n] < w[\mathsf{SA}[j], n]$ for every $1 \leq i < j \leq n$. The *Inverse Suffix Array* (ISA) is the inverse of SA, i.e. $\mathsf{ISA}[i] = j$ if and only if $\mathsf{SA}[j] = i$. Analogously, let CA be the *Conjugate Array* of a word w, defined as:

$$\mathsf{CA}[i] = j \text{ if } w_i = w[j, n]w[1, j-1],$$

where w_i is the ith rotation in lexicographical order.

Given two words $u, v \in \Sigma^*$, let $\ell cp(u, v)$ be the the longest common prefix between u and v, that is $\ell cp(u, v) = u[1, |\ell cp(u, v)|] = v[1, |\ell cp(u, v)|]$, but $u[|\ell cp(u, v)|+1] \neq v[|\ell cp(u, v)|+1]$ (assuming $\ell cp(u, v) < \min\{|u|, |v|\}$). The *Longest Common Prefix Array* (LCP) of $w \in \Sigma^n$ is an array of length n such that $\mathsf{LCP}[1] = 0$ and $\mathsf{LCP}[i] = |\ell cp(w[\mathsf{SA}[i-1], n], w[\mathsf{SA}[i], n))|$, where $1 < i < n$. Analogously, we denote by LCP_c the *circular Longest Common Prefix Array*, defined as follows:

$$\mathsf{LCP}_c[i] = \begin{cases} 0 & \text{if } i = 0 \\ |\ell cp(w_{i-1}, w_i)| & \text{otherwise} \end{cases} .$$

A *string attractor* of a word $w \in \Sigma^n$ is a set of γ positions $\Gamma = \{p_1, \ldots, p_\gamma\}$ such that every factor $w[i, j]$ has an occurrence $w[i', j'] = w[i, j]$ with $p_k \in [i', j']$, for some $k \in [1, \gamma]$ [1]. We denote by $\gamma^*(w)$ the size of a smallest string attractor of a word w. Furthermore, we say that an occurrence $w[i, j]$ *crosses* a position $p \in \Gamma$ if $p \in [i, j]$, and symmetrically that p is *crossed by* $w[i, j]$.

Throughout the paper, we will show examples of the algorithms applied on the family of *finite Fibonacci words*, which can be defined recursively as follows: $f_0 = \mathtt{b}$, $f_1 = \mathtt{a}$, and $f_{i+1} = f_i \cdot f_{i-1}$, for all integer $i > 1$.

3. Circular String Attractors

Analogously to the definition of string attractor, Mantaci et al. [2] extended such a notion in order to capture factors that occur on the boundaries of a word.

Definition 1 *Let $w \in \Sigma^n$, for some $n > 0$. A set $\Gamma_c = \{j_1, j_2, \ldots j_{\gamma_c}\} \subseteq [1, n]$ is a* circular string attractor *of w if each circular factor of w has at least a circular occurrence that crosses a position of Γ_c. Moreover, we denote with $\gamma_c^*(w)$ the size of a smallest circular string attractor of the word w.*

Example 1 *Let $w = a\underline{b}bb\underline{c}aa\underline{a}caaa$ be a word over the alphabet $\Sigma = \{a, b, c\}$. The set $\Gamma = \{2, 5, 8\}$ is a string attractor for w, since it covers any of its factors, but it is not a circular string attractor since the circular factor (in blue) $caaaa$ escapes from it. On the other hand, the set $\Gamma_c = \{1, 4, 9\}$ is a circular string attractor for $w = \underline{a}bb\underline{b}caaa\underline{c}aaa$ but it is not a string attractor. In fact, the factor aaa (in blue), fully contained in w, is covered only if we consider its circular occurrence crossing position 1.*

From the definition, it easily follows that the sizes γ_c^* of smallest circular string attractors of conjugate words are equal [2].

The following lemma shows not only that $\gamma_c^*(w) = \gamma_c^*(w^m)$, for all $w \in \Sigma^*$ and integer $m > 0$, but that also the structures of the respective circular string attractors are related.

Lemma 1 *Let $w \in \Sigma^*$ be a finite word and $\Gamma_c = \{p_1, p_2, \ldots, p_{\gamma_c}\} \subseteq [1, |w|]$. Then, Γ_c is a circular string attractor of w if and only if, for all $m > 1$ and for all $d \in [0, m-1]^{\gamma_c}$, the set $\Gamma' = \bigcup_{i \in [1, \gamma_c]} \{p_i + d_i|w|\}$ is a circular string attractor of w^m.*

Proof *For the first direction, note that since the word w^m has period $|w|$, all circular factors of length at most $|w|$ which cross the position p_i for some $i \in [1, \gamma_c]$ have another occurrence crossing the position $p_i + d_i|w|$. For all the circular factors of w^m of length greater than $|w|$, for the same argument we can find (at least) m circular occurrences through w^m at distance $|w|$, covering the*

$$\begin{array}{ccccccccccccc}
\mathtt{a} & \mathtt{a} & \mathtt{b} & \mathtt{a} & \underline{\mathtt{a}} & \underline{\mathtt{b}} & \mathtt{a} & \mathtt{b} & \mathtt{a} & \mathtt{a} & \mathtt{b} & \mathtt{a} & \mathtt{b} \\
\mathtt{a} & \mathtt{a} & \mathtt{b} & \mathtt{a} & \mathtt{b} & \mathtt{a} & \mathtt{a} & \mathtt{b} & \mathtt{a} & \underline{\mathtt{a}} & \underline{\mathtt{b}} & \mathtt{a} & \mathtt{b} \\
\mathtt{a} & \underline{\mathtt{a}} & \underline{\mathtt{b}} & \mathtt{a} & \mathtt{b} & \mathtt{a} & \mathtt{a} & \mathtt{b} & \mathtt{a} & \mathtt{b} & \mathtt{a} & \mathtt{a} & \mathtt{b} \\
\mathtt{a} & \mathtt{b} & \mathtt{a} & \mathtt{a} & \mathtt{b} & \mathtt{a} & \underline{\mathtt{a}} & \underline{\mathtt{b}} & \mathtt{a} & \mathtt{b} & \mathtt{a} & \mathtt{a} & \mathtt{b} \\
\mathtt{a} & \mathtt{b} & \mathtt{a} & \mathtt{a} & \mathtt{b} & \mathtt{a} & \mathtt{b} & \mathtt{a} & \mathtt{a} & \mathtt{b} & \mathtt{a} & \underline{\mathtt{a}} & \underline{\mathtt{b}} \\
\mathtt{a} & \mathtt{b} & \mathtt{a} & \underline{\mathtt{a}} & \underline{\mathtt{b}} & \mathtt{a} & \mathtt{b} & \mathtt{a} & \mathtt{a} & \mathtt{b} & \mathtt{a} & \mathtt{b} & \mathtt{a} \\
\mathtt{a} & \mathtt{b} & \mathtt{a} & \mathtt{b} & \mathtt{a} & \mathtt{a} & \mathtt{b} & \mathtt{a} & \underline{\mathtt{a}} & \underline{\mathtt{b}} & \mathtt{a} & \mathtt{b} & \mathtt{a} \\
\underline{\mathtt{a}} & \underline{\mathtt{b}} & \mathtt{a} & \mathtt{b} & \mathtt{a} & \mathtt{a} & \mathtt{b} & \mathtt{a} & \mathtt{b} & \mathtt{a} & \mathtt{a} & \mathtt{b} & \mathtt{a} \\
\mathtt{b} & \mathtt{a} & \mathtt{a} & \mathtt{b} & \mathtt{a} & \underline{\mathtt{a}} & \underline{\mathtt{b}} & \mathtt{a} & \mathtt{b} & \mathtt{a} & \mathtt{a} & \mathtt{b} & \mathtt{a} \\
\mathtt{b} & \mathtt{a} & \mathtt{a} & \mathtt{b} & \mathtt{a} & \mathtt{b} & \mathtt{a} & \mathtt{a} & \mathtt{b} & \mathtt{a} & \underline{\mathtt{a}} & \underline{\mathtt{b}} & \mathtt{a} \\
\mathtt{b} & \mathtt{a} & \underline{\mathtt{a}} & \underline{\mathtt{b}} & \mathtt{a} & \mathtt{b} & \mathtt{a} & \mathtt{a} & \mathtt{b} & \mathtt{a} & \mathtt{b} & \mathtt{a} & \mathtt{a} \\
\mathtt{b} & \mathtt{a} & \mathtt{b} & \mathtt{a} & \mathtt{a} & \mathtt{b} & \mathtt{a} & \underline{\mathtt{a}} & \underline{\mathtt{b}} & \mathtt{a} & \mathtt{b} & \mathtt{a} & \mathtt{a} \\
\underline{\mathtt{b}} & \mathtt{a} & \mathtt{b} & \mathtt{a} & \mathtt{a} & \mathtt{b} & \mathtt{a} & \mathtt{b} & \mathtt{a} & \mathtt{a} & \mathtt{b} & \mathtt{a} & \underline{\mathtt{a}}
\end{array}$$

Figure 1: Matrix of the sorted rotations for the finite Fibonacci word $f_6 = \mathtt{abaababaaba}\underline{\mathtt{ab}}$. The underlined positions correspond to the positions of a circular string attractor $\Gamma_c = \{12, 13\}$. We underline in the other rotations the corresponding positions.

whole word, and therefore these factors can be moved to every other occurrence of w where a position from Γ' falls.

For the second implication, by hypothesis the set Γ_c is a circular string attractor of w^m. Symmetrically to the previous direction, since the set of all circular factors of w^m of length at most $|w|$ corresponds to the set $\mathcal{C}(w)$, and each of these circular factors from $\mathcal{C}(w)$ is fully covered from the elements in Γ_c, the thesis follows.

4. Checking the Circular Attractor Property in Linear Time

Given a word w of length $n > 0$, let us consider the matrix $\mathcal{M}(w) = \{w_1, \ldots, w_n\}$ of the rotations of w sorted in lexicographical order. For every factor $u \in \mathcal{C}(w)$, we denote with $\mathcal{I}_u$ the set of rotations from $\mathcal{M}(w)$ starting with u, taken in lexicographical order. In order to give a high-level view of how the algorithm works, let us consider the finite Fibonacci word $f_6 = \mathtt{abaababaabaab}$. In Figure 1, it is shown the matrix of sorted rotations of f_6 and, for each rotation, the positions of a circular string attractor described in [2]. One can verify that, for each circular factor $u \in \mathcal{C}(f_6)$, the rotations having u as prefix are consecutive, and at least one of these prefixes crosses a position of the circular string attractor. Note that this is not a matter of chance, since each occurrence of every circular factor is a prefix of a rotation.

For each $u \in \mathcal{C}(w) \setminus \{\varepsilon\}$, let ℓ_u, r_u, with $1 \leq \ell_u \leq r_u \leq n$, be the indices such that $\mathcal{I}_u = \{w_{\ell_u}, w_{\ell_u+1}, \ldots, w_{r_u}\}$. The following lemma summarises how to detect the indices ℓ_u and r_u from the LCP$_c$ array.

Lemma 2 *Let $w \in \Sigma^n$, for some $n > 0$. The following hold:*

1. $\ell_u = 1$ *or* $LCP_c[\ell_u] < |u|$;
2. $r_u = n$ *or* $LCP_c[r_u + 1] < |u|$;

3. *$LCP_c[k] \geq |u|$, for all $k \in [\ell_u + 1, r_u]$.*

Proof *For case 1., suppose $\ell_u > 1$. By contradiction, if $LCP_c[\ell_u] \geq u$, then also the rotation w_{ℓ_u+1} has u as prefix, which is a contradiction by hypothesis on ℓ_u.*

Case 2. is treated symmetrically.

Case 3. follows by observing that all the rotations in $\mathcal{I}_u$ are consecutive and share the same prefix of length $|u|$. □

Given an ordered set $\Gamma_c = \{p_1, p_2, \ldots, p_{\gamma_c}\}$ and a word $w \in \Sigma^n$, let $\mathrm{succ}_c \in \{0, 1, \ldots, n-1\}^n$ be the array of circular distances of each position $i \in [1, n]$ of w to the next position p in Γ_c, that is:

$$\mathrm{succ}_c[i] = \begin{cases} p_1 - i & \text{if } 1 \leq i \leq p_1 \\ p_{j+1} - i & \text{if } p_j < i \leq p_{j+1}, \text{ for all } j \in [1, \gamma_c - 1] \\ (n - i) + p_1 & \text{if } p_{\gamma_c} < i. \end{cases}$$

Example 2 *Let us consider the word $w = \mathtt{ab\underline{b}a\underline{ba}a}$ and the set $\Gamma_c = \{3, 5, 6\}$ (the underlined positions in w). Then, $succ_c = [2, 1, 0, 1, 0, 0, 3]$.*

By Lemma 1, we know that a set Γ_c is a circular string attractor for the word w^n, where $n > 1$, if and only if $\Gamma'_c = \bigcup_{p \in \Gamma_c} \{(p - 1 \mod |w|) + 1\}$ is a circular string attractor of w. Thus, we can assume that w is primitive, otherwise we can find its root u in linear time and check whether the set Γ'_c obtained as just described is a circular string attractor of u.

Lemma 3 *Let $w \in \Sigma^n$ be a primitive word, and let $\Gamma \subseteq [1, n]$ be a set of positions in w, for some $n > 0$. Then, Γ_c is a circular string attractor of w if and only if, for all $u \in \mathcal{C}(w)$, there exists $i \in [\ell_u, r_u]$ such that $succ_c[CA[i]] < |u|$.*

Proof *For the first implication, if Γ_c is a circular string attractor, then for each $u \in \mathcal{C}(w)$ there exists at least one occurrence that is crossed by a position $p \in \Gamma_c$. Let $i \in [\ell_u, r_u]$ be the index of the rotation starting with such an occurrence of u. Thus, if we project the position from Γ_c in the ith rotation, such a position falls at most at distance $|u|$, and therefore $succ_c[CA[i]]$ is at most $|u| - 1$.*

The other direction is treated symmetrically. In fact, by contradiction, if for all $i \in [\ell_u, r_u]$ it holds that $succ_c[CA[i]] \geq |u|$, then none of the occurrences of u crosses a position in Γ_c, and therefore it can not be a circular string attractor; contradiction. □

Thus, in order to check whether a set Γ_c is a circular string attractor, we need to check if, for all $u \in \mathcal{C}(w)$, there exists $i \in [\ell_u, r_u]$ such that $\mathrm{succ}_c[\mathrm{CA}[i]] < |u|$. In Algorithm 1 we describe the designed procedure. We store in a stack S the ranges of lengths of the circular factors left to cover, and proceeding by comparing in order the LCP_c with the succ_c array. We use Lemma 2 to understand if we are still in the range $[\ell_u, r_u]$ without even knowing u, but just by using the LCP_c array (line 8). Note that this is legit since for all $u, v \in \Sigma^*$ it holds that $[\ell_{uv}, r_{uv}] \subseteq [\ell_u, r_u]$, i.e., we can not leave the range $[\ell_u, r_u]$ before checking if the factor uv is covered within $[\ell_{uv}, r_{uv}]$. The consecutive lengths of the circular factors left to cover are inserted as a pair (s, e) in S.

We can then obtain the following:

Algorithm 1: Algorithm for checking if a set Γ_c is a circular string attractor of a word w

```
1  $S \leftarrow$ empty stack
2  $\mathsf{succ}_c \leftarrow computeCircSucc(w, \Gamma)$
3  $\mathsf{CA} \leftarrow computeConjugateArray(w)$
4  $\mathsf{LCP}_c \leftarrow computeCircularLongestCommonPrefixArray(w)$
5  for $i \in [1, n]$ do
6      if $S$ is not empty then
7          $(s, e) \leftarrow S.pop()$
8          if $LCP_c[i] < e$ then
9              return false
10         else
11             if $LCP_c[i] \leq succ_c[CA[i]]$ then
12                 if $e = LCP_c[i]$ then
13                     $S.push((s, \mathsf{succ}_c[\mathsf{CA}[i]]))$
14                 else
15                     $S.push((s, e))$
16                     if $LCP_c[i] < succ_c[CA[i]]$ then $S.push((\mathsf{LCP}_c[i] + 1, \mathsf{succ}_c[\mathsf{CA}[i]]))$
18             else
19                 while $S$ is not empty $\wedge$ $succ_c[CA[i]] \leq s$ do
20                     $(s, e) \leftarrow S.pop()$
21                 if $s < succ_c[CA[i]]$ then
22                     $S.push((s, \min\{e, \mathsf{succ}_c[\mathsf{CA}[i]]\}))$
23     else
24         if $LCP_c[i] < succ_c[CA[i]]$ then $S.push((\mathsf{LCP}_c[i] + 1, \mathsf{succ}_c[\mathsf{CA}[i]]))$
25 if $S$ is empty then
26     return true
27 else
28     return false
```

Theorem 1 *Given a primitive word $w \in \Sigma^n$ and a set $\Gamma_c \subseteq [1, n]$, Algorithm 1 checks whether or not the set Γ_c is a circular string attractor for w. Moreover, it runs in $O(n)$ time using $O(n \log n)$ bits of space.*

Proof *The Conjugate Array CA can be computed in linear time [8]. The LCP$_c$ array can be computed in linear time from CA [9, 10]. The main loop (line 5) is executed at most n times. Further, the inner loop in line 19 where we empty the stack S can be executed at most $|S|$ times in total. Since we add at most one pair in S at each iteration of the main loop, it holds that the number of elements in the stack is $|S| = O(n)$, occupying at most $O(n \log n)$ bits of space. Thus, Algorithm 1 works in $O(n)$ time using $O(n \log n)$ bits of space and the thesis follows.* □

Example 3 *In Figure 2, we show the steps of Algorithm 1 applied to the* 5*th finite Fibonacci word* $f_5 = \texttt{abaababa}$ *with the set* $\Gamma_c = \{4, 5\}$*. The matrix of the sorted rotations is shown to give to the reader a graphical interpretation of the procedure, however recall that we do not use it.*

The stack S *is initially empty, so at the iteration* $i = 1$ *we start to fill it with the lengths of the prefixes of the first rotation in lexicographical order that need to cross a position from* Γ_c*. Since* $succ_c[CA[1]] = 4$*, this means that all prefixes of the first rotation with length greater than* 4 *cross a position from* Γ_c*. On the other hand, we can not say anything yet on the prefixes from length* 1 *to* 4*, which are* a*,* aa*,* aab*, and* aaba*. In fact, since* $LCP_c[1] = 0 < 4 = succ_c[CA[1]]$*, we push into the stack* S *the pair* $(1, 4)$ *(line 24), as displayed in Subfigure 2a.*

On the iteration $i = 2$*, we pop from the stack the ranges of lengths* $(1, 4)$*, and we compare the maximum (*4*) with* $LCP_c[2]$*, since we want to check whether there is another occurrence of all the factors represented in the stack. Since* $LCP_c[2] = 4$*, we have another rotation with prefix* aaba*, and therefore such a rotation starts by* a*,* aa*, and* aab *as well. In Subfigure 2b, since* $succ_c[CA[2]] = 1$*, analogously to the previous case all prefixes longer than* 1 *cross a position from* Γ_c*, and therefore only* a *is left to cover, and we insert in the stack the range* $(1, 1)$ *(line 22).*

Then, at the iteration $i = 3$ *(Subfigure 2c), the prefix* a *still occurs (*$LCP_c[3] = 1$*), but it does not cross a position from* Γ_c *(*$succ_c[CA[3]] = 6$*), so we extend the range from* $(1, 1)$ *to* $(1, 6)$ *and push it in* S *(line 13), that is we keep track of the factors* a*,* ab*,* aba*,* abaa*,* abaab*, and* abaaba*.*

The algorithm proceeds as shown in Subfigure 2d, by shrinking the range $(1, 6)$ *to* $(1, 3)$ *(line 22). At the following iteration shown in Subfigure 2e the stack is emptied for the first time, since* $succ_c[CA[5]] = 0$ *and the condition of line 21 is not met.*

We keep following the procedure for the iterations 6*,* 7*, and* 8*, respectively shown in Subfigures 2f, 2g, and 2h. Since at the end of the main loop the stack is empty, each circular factor crosses at least a position from* Γ_c*, and therefore we return* ***true****.*

Remark 1 *Given a pair* (s, e) *in* S*, every pair* (s', e') *that goes on top of* (s, e) *must verify that* $s' > e$*, since if* $s' = e$ *then* (s, e) *is replaced with* (s, e')*. Thus, in the worst-case scenario* S *would contain the pairs* $(1, 1), (3, 3), \ldots, (\left\lfloor \frac{n}{2} \right\rceil, \left\lfloor \frac{n}{2} \right\rceil)$*, that is* S *requires at most* $n \log n$ *bits of space. Overall, Algorithm 1 requires up to* $4n \log n$ *bits of space to run.*

5. Complexity of circular-attractor

In their original work, Kempa and Prezza [1] have formulated the following decision problem

$$\text{attractor} = \{\langle w, p \rangle : w \text{ has a string attractor of size } \leq p\}.$$

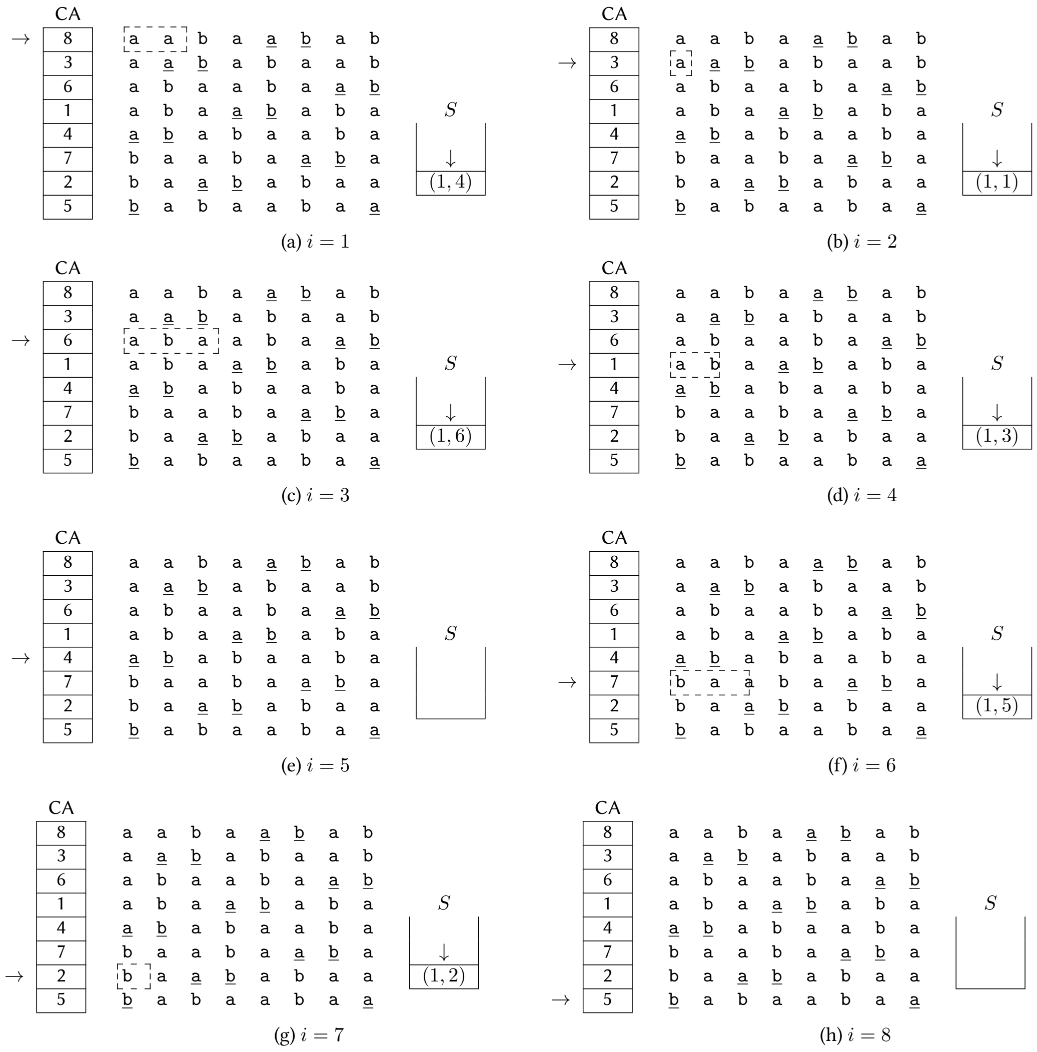
In the same work, they proved that this problem is NP-complete. Here we define the analogous problem extended to the notion of circular string attractor:

$$\text{circular-attractor} = \{\langle w, p \rangle : w \text{ has a circular string attractor of size } \leq p\}.$$

The following lemma shows a reduction to the analogous problem on circular string attractors.

Lemma 4 *Given an integer* $n > 0$*, let* $w \in \Sigma^n$ *be a finite word and let* $\$ \notin \Sigma$*. A set* Γ *is a string attractor for* w *if and only if* $\Gamma \cup \{n + 1\}$ *is a circular string attractor for* $w\$$

$i =$	1	2	3	4	5	6	7	8
$f_5 =$	a	b	a	$\underline{\text{a}}$	$\underline{\text{b}}$	a	b	a
$\mathsf{succ}_c =$	3	2	1	0	0	6	5	4
$\mathsf{LCP}_c =$	0	4	1	6	3	0	5	2



(a) $i = 1$

(b) $i = 2$

(c) $i = 3$

(d) $i = 4$

(e) $i = 5$

(f) $i = 6$

(g) $i = 7$

(h) $i = 8$

Figure 2: Running example of Algorithm 1 on the word $f_5 = \mathsf{abaababa}$. The underlined positions in f_5 correspond to the positions of the circular string attractor Γ_c, and the corresponding succ_c and LCP_c arrays are shown right above. Each subfigure shows one of the 8 iterations of the algorithm, and the stack S at the end of the iteration. The dashed boxes surround the prefixes of the current rotation for which we have not found an occurrence crossing a position in Γ_c. The lengths of these prefixes correspond to the ranges in S.

Proof *Let $\mathcal{C}_\$(w\$)$ denote the set of all circular factors of $w\$$ containing the letter $\$$. One can observe that $\mathcal{C}(w\$) = \mathcal{F}(w) \cup \mathcal{C}_\$(w\$)$. Indeed, $\mathcal{F}(w) \cap \mathcal{C}_\$(w_\$) = \emptyset$, and the position $\{n+1\}$ is crossed by all and only circular factors from $\mathcal{C}_\$(w_\$)$. Thus, the factors to cover in w are the same as the circular factors in $\mathcal{C}(w\$) \setminus \mathcal{C}_\$(w\$)$, and since they are located in the same positions in both words the thesis follows.* □

From the reduction above, we can deduce the computational complexity of the circular-attractor problem.

Theorem 2 *The circular-attractor problem is NP-complete.*

Proof *By Theorem 1, each solution for the circular-attractor problem can be checked in polynomial time and space, and therefore circular-attractor $\in$ NP. Furthermore, Lemma 4 shows how to reduce an instance $\langle w, p\rangle$ for the problem attractor to circular-attractor by replacing w and p with $w\$$ and $p+1$ respectively, since for each string attractor Γ of w the set $\Gamma \cup \{n+1\}$ is a circular string attractor of $w\$$.* □

6. Novel Algorithm for Checking the Attractor Property

If we consider the same problem on classical string attractors, Kempa et al. presented two algorithms for checking whether a set Γ is a string attractor for a word $w \in \Sigma^n$ [7]. The first algorithm uses optimal $O(n \log \sigma)$ bits of space and operates in $O(n \log^\epsilon n)$ time, for any constant $\epsilon > 0$. To reach the linear time, Kempa et al. proposed another algorithm requiring $O(n(\log \sigma + \log n))$ bits of space. Both algorithms are based on suffix trees and other data structures supporting range-minimum queries, for which the implementation is not trivial in all programming languages.

Here we present a novel algorithm taking the same time and space complexities as Algorithm 1. The algorithm is constructed from the strategies developed in Algorithm 1 and from the equivalence of Lemma 4. Recall that appending a $\$$ smaller than any other symbol in Σ implies that $\mathsf{CA} = \mathsf{SA}$ and $\mathsf{LCP}_c = \mathsf{LCP}$. We can then derive the following Theorem.

Theorem 3 *Given a word $w \in \Sigma^n$ and a set $\Gamma \subseteq [1, n]$, Algorithm 2 checks whether or not the set Γ is a string attractor for w in $O(n)$ time using $O(n \log n)$ bits of space.*

Proof *The correctness of Algorithm 2 is derived from its equivalence in Lemma 4 with Algorithm 1. The Suffix Array SA, the LCP array, and the succ array can be computed in $O(n)$ time using $O(n \log n)$ bits of space. Note that, unlike Algorithm 1, Algorithm 2 takes in input also words that are not primitive, since by appending the letter $\$ \notin \Sigma$ every word becomes primitive. Since Algorithm 2 takes the same time and space as Algorithm 1, the thesis follows.* □

Example 4 *Let us consider the word $f' =$ `ababaaba`, that is a rotation of the word f_5 from Example 3, and let us consider the set of positions $\Gamma = \{7, 8\}$. In Figure 3, the iterations of Algorithm 2 with f' and Γ in input are shown. Recall that we compute the LCP, succ, and SA arrays for the word $f'\$$, and we extend Γ with the position of the $\$$, i.e. $\Gamma = \{7, 8, 9\}$.*

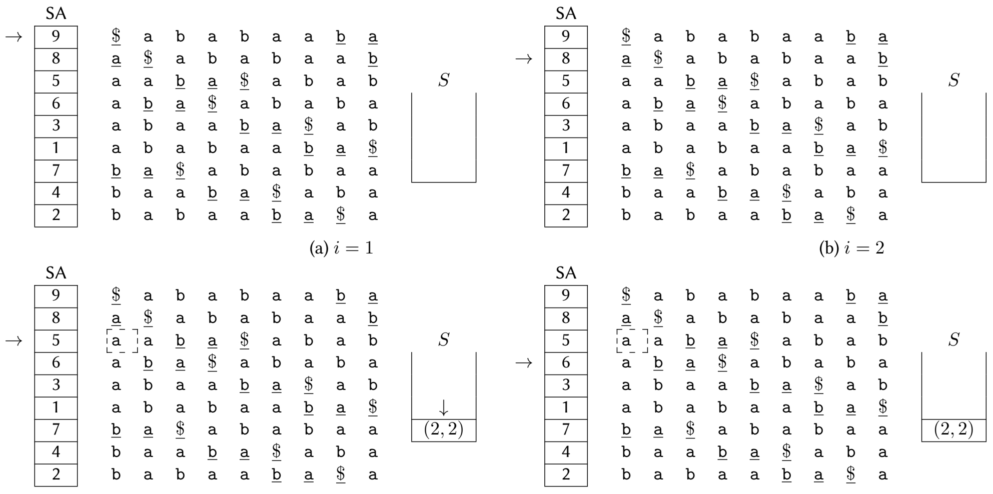
Algorithm 2: Algorithm for checking if a set Γ is a string attractor for a word w

```
1  $S \leftarrow$ empty stack
2  $\Gamma \leftarrow \Gamma \cup \{n+1\}$
3  SA $\leftarrow computeSuffixArray(w\$)$
4  succ $\leftarrow computesucc(w\$, \Gamma)$
5  LCP $\leftarrow computeLongestCommonPrefixArray(w\$)$
6  for $i \in [1, n]$ do
7      if $S$ is not empty then
8          $(s, e) \leftarrow S.pop()$
9          if $LCP[i] < e$ then
10             return false
11         else
12             if $LCP[i] \leq succ[SA[i]]$ then
13                 if $e = LCP[i]$ then
14                     $S.push((s, \text{succ}[\text{SA}[i]]))$
15                 else
16                     $S.push((s, e))$
17                     if $LCP[i] < succ[SA[i]]$ then $S.push((\text{LCP}[i] + 1, \text{succ}[\text{SA}[i]]))$
18             else
19                 while $S$ is not empty $\wedge succ[SA[i]] \leq s$ do
20                     $(s, e) \leftarrow S.pop()$
21                 if $s < succ[SA[i]]$ then
22                     $S.push((s, \min\{e, \text{succ}[\text{SA}[i]]\}))$
23     else
24         if $LCP[i] < succ[SA[i]]$ then $S.push((\text{LCP}[i] + 1, \text{succ}[\text{SA}[i]]))$
25 if $S$ is empty then
26     return true
27 else
28     return false
```

As shown in Subfigure 3a, at first the stack is empty and the condition of line 24 is not met, since $LCP[1] = succ[SA[1]] = 0$, and therefore nothing is added into S. The same procedure occurs at the following iteration, in Subfigure 3b.

Since at the third iteration the stack is still empty, each factor that is prefix of the first two rotations has an occurrence crossing at least a position in Γ. Since this time $1 = LCP[3] < succ[SA[3]] = 2$, we add to the stack in line 24 only the factors that have not occurred yet (i.e. of length at least $LCP[3] + 1$) and that are not crossing a position in Γ (i.e. of length at most $succ[SA[3]]$), and therefore we add the range $(2, 2)$. As shown in Subfigure 3c, such a range corresponds to the factor aa.

$i =$	1	2	3	4	5	6	7	8	9
$f'\$ =$	a	b	a	b	a	a	$\underline{\text{b}}$	$\underline{\text{a}}$	$\underline{\$}$
succ =	6	5	4	3	2	1	0	0	0
LCP =	0	0	1	1	3	3	0	2	2



(a) $i = 1$

(b) $i = 2$

(c) $i = 3$

(d) $i = 4$

Figure 3: Running example of Algorithm 2 on the word $f' =$ ababaaba\$. The underlined positions in f' correspond to the positions of a set Γ that we want to check whether it is a string attractor for f'. The corresponding succ and LCP arrays are shown right above. Each subfigure shows one of the 4 iterations of the algorithm, and the status of the stack S at the end of the iteration. The dashed boxes surround the prefixes of the current rotation for which we have not found an occurrence crossing a position in Γ. The lengths of these prefixes correspond to the ranges in S.

Finally, in Subfigure 3d, one can see that the factor aa *does not occur as prefix in the following rotations. In fact, at the iteration $i = 4$ Algorithm 2 checks in line 9 whether the factor that we are looking for occurs again as prefix by comparing LCP$[4] = 1$ with the maximum value from the top of the stack. Since the condition is not met, the algorithm returns* ***false****, i.e. the factor* aa *is not crossed by any position in Γ, and therefore Γ is not a string attractor.*

7. Conclusions

In this work, we have shown an easy reduction from the attractor decision problem to its circular version, leading to an NP-completeness for circular-attractor. We have also presented the first algorithm in literature that checks whether a set Γ_c is a circular string attractor for a word $w \in \Sigma^n$, operating in $O(n)$ time and $O(n \log n)$ bits of space. Furthermore, given the reduction above mentioned and the equivalence between the order of the rotations and order of the suffixes for words of the type $w\$$, we have presented a new algorithm for checking the attractor property of a set by using the suffix and the Longest Common Prefix Arrays, using the same time and space bound as the previous one. With respect to the solutions proposed by Kempa et al., the algorithm here proposed is independent from the size of the alphabet.

References

[1] D. Kempa, N. Prezza, At the roots of dictionary compression: string attractors, in: STOC 2018, ACM, 2018, pp. 827–840.

[2] S. Mantaci, A. Restivo, G. Romana, G. Rosone, M. Sciortino, A combinatorial view on string attractors, Theor. Comput. Sci. 850 (2021) 236–248.

[3] K. Kutsukake, T. Matsumoto, Y. Nakashima, S. Inenaga, H. Bannai, M. Takeda, On repetitiveness measures of Thue-Morse words, in: SPIRE, volume 12303 of *Lect. Notes Comput. Sc.*, Springer, 2020, pp. 213–220.

[4] L. Schaeffer, J. Shallit, String attractors for automatic sequences, CoRR abs/2012.06840 (2021). arXiv:2012.06840.

[5] A. Restivo, G. Romana, M. Sciortino, String attractors and infinite words, in: LATIN, volume 13568 of *Lecture Notes in Computer Science*, Springer, 2022, pp. 426–442.

[6] F. Gheeraert, G. Romana, M. Stipulanti, String attractors of fixed points of k-bonacci-like morphisms, in: WORDS, volume 13899 of *Lecture Notes in Computer Science*, Springer, 2023, pp. 192–205.

[7] D. Kempa, A. Policriti, N. Prezza, E. Rotenberg, String Attractors: Verification and Optimization, in: ESA, volume 112 of *Leibniz International Proceedings in Informatics (LIPIcs)*, Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, 2018, pp. 52:1–52:13.

[8] C. Boucher, D. Cenzato, Z. Lipták, M. Rossi, M. Sciortino, Computing the original ebwt faster, simpler, and with less memory, in: SPIRE, volume 12944 of *Lecture Notes in Computer Science*, Springer, 2021, pp. 129–142.

[9] T. Kasai, G. Lee, H. Arimura, S. Arikawa, K. Park, Linear-time longest-common-prefix computation in suffix arrays and its applications, in: CPM, volume 2089 of *Lecture Notes in Computer Science*, Springer, 2001, pp. 181–192.

[10] J. Kärkkäinen, G. Manzini, S. J. Puglisi, Permuted longest-common-prefix array, in: CPM, volume 5577 of *Lecture Notes in Computer Science*, Springer, 2009, pp. 181–192.