





On Suffix Tree Detection

Amihood Amir¹ , Eitan Kondratovsky² , and Avivit Levy³  

¹ Department of Computer Science, Bar-Ilan University, Ramat-Gan, Israel

amir@esc.biu.ac.il

² Cheriton School of Computer Science, Waterloo University, Waterloo, Canada

e2kondra@uwaterloo.ca

³ Software Engineering Department, Shenkar College of Engineering and Design,
Ramat-Gan, Israel

avivitlevy@shenkar.ac.il

<https://u.cs.biu.ac.il/~amir/>, <https://u.cs.biu.ac.il/~kondrae>

Abstract. A suffix tree is a fundamental data structure for string processing and information retrieval, however, its structure is still not well understood. The *suffix trees reverse engineering problem*, which its research aims at reducing this gap, is the following. Given an ordered rooted tree T with unlabeled edges, determine whether there exists a string w such that the unlabeled-edges suffix tree of w is isomorphic to T . Previous studies on this problem consider the relaxation of having the suffix links as well as assume a binary alphabet. This paper is the first to consider the *suffix tree detection problem*, in which the relaxation of having suffix links as input is removed. We study suffix tree detection on two scenarios that are interesting per se. We provide a suffix tree detection algorithm for *general alphabet periodic strings*. Given an ordered tree T with n leaves, our detection algorithm takes $O(n + |\Sigma|^p)$ -time, where p is the *unknown in advance length* of a period that repeats at least 3 times in a string S having a suffix tree structure identical to T , if such S exists. Therefore, it is a polynomial time algorithm if p is a constant and a linear time algorithm if, in addition, the alphabet has a sub-linear size. We also show some necessary (but insufficient) conditions for *binary alphabet general strings* suffix tree detection. By this we take another step towards understanding suffix trees structure.

Keywords: Suffix tree · Reverse engineering · Suffix tree detection · Periodic string

1 Introduction

A suffix tree is a fundamental data structure for string processing and information retrieval being one of the most well-known and widely used text indexing structures. It provides a linear space full-text index of a given string and has played a central role in combinatorial pattern matching and its applications. A multitude of important problems can efficiently be solved using suffix trees [3, 15].

Partly supported by ISF grant 1475/18 and BSF grant 2018141.

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2023

F. M. Nardini et al. (Eds.): SPIRE 2023, LNCS 14240, pp. 14–27, 2023.

https://doi.org/10.1007/978-3-031-43980-3_2

Despite their essential role, the structure of suffix trees is still not well understood. For instance, only almost forty years after this data structure was first introduced by Peter Weiner [26], it was proved that each internal edge in a suffix tree can contain at most one implicit suffix node [5]. For suffix arrays, which are a known alternative data structure to suffix trees, several characterization theorems are known (e.g. [4, 16, 20, 23]). For example, Kucherov et al. [20] present a bijective characterization of suffix array permutations obtained from a characterization of Burrows-Wheeler arrays. Considering the *suffix trees reverse engineering problem*, which was first presented by [18] and studied also by [6, 24], is a step towards closing this gap in the understanding of suffix trees structure.

The idea of *reverse engineering* has been studied for several data structures. For example, Franek et al. [11] verify a border array in linear time; I et al. [17] verify and enumerate a parameterized border array; Duval et al. [9] validate string automata; Bannai et al. [4] infer strings from graphs; Clément et al. [7] reverse engineer prefix tables; Gawrychowski et al. [12] validate the KMP failure function; Crochemore et al. [8] study cover array string recognition; Kärkkäinen et al. [19] infer a string from the LCP array; Nakashima et al. [22] infer a string from the Lyndon factorization and Gawrychowski et al. [13] prove some general conditions that allow string reconstruction (e.g., maximal palindromes). There are some known negative reverse engineering results, such as Gelle and Iván [14] who showed that recognizing Union-Find trees is NP-complete.

The *suffix trees reverse engineering problem* is the following. Given an ordered rooted tree T with unlabeled edges, determine whether there exists a string w such that the unlabeled-edges suffix tree of w is isomorphic to T . I et al. [18] emphasize that this problem is very challenging due to the following reasons:

1. The length of each edge string is not given;
 2. The mapping from strings to edge-unlabeled suffix trees is not injective.
- Therefore, only relaxed versions of the problem had been solved until now.

As a first step towards solving the problem, I et al. [18] restrict the problem to a binary alphabet. In addition, they assume that suffix links of inner nodes are given as input. Under these conditions, they solve the suffix trees reverse engineering problem in linear time in the size of the input tree T . All other attempts also assume that the suffix links are given. Starikovskaya and Vildhøj [24] prove some new properties of suffix trees and show how to reverse engineer implicit suffix trees (where there is no special end-of-string symbol). Casaux and Rivals [6] assume the existence of leaf suffix links (where each leaf points to the next longer leaf) besides the existence of internal nodes suffix links. Under these assumptions they reverse engineer suffix trees over general alphabets in linear time.

Removing the relaxation of having the suffix links is quite a challenge posing additional difficulties on the suffix tree structure analysis. For example:

1. It changes the degrees of freedom for the inner nodes suffix links references from 1 to $O(n)$, potentially, where n is the number of the suffix tree leaves, as the number of inner nodes is $O(n)$.

2. Even for a binary alphabet, the ordered suffix tree structure does not necessarily obey character-symmetry laws, i.e., by flipping a binary string characters, one is not guaranteed to have a suffix tree with an isomorphic structure.

Therefore, the task of studying ordered suffix trees structure (even on binary strings) where the suffix links *are not given as input*, is a natural next step in the study of the structure of suffix trees, but also requires to cope with such difficulties.

In this paper, we consider the reverse engineering problem, in which the relaxation of having the suffix links as input is removed. Formally,

Definition 1. The Suffix Tree Detection problem *is the following*:

Input: An ordered rooted tree T

Output: YES, if there exists a string S such that its unlabeled-edges ordered suffix tree is isomorphic to T ,
NO, otherwise.

We study suffix tree detection on two scenarios that are interesting per se: (1) *general alphabet periodic strings* and (2) *binary alphabet general strings*. The conditions on these two classes of strings considered in this paper are indeed restrictive. Nevertheless, these classes are rich enough to retain an inherent difficulty of removing the relaxation of suffix links availability as input.

Paper Contribution. The main contributions of this paper are:

- Being the first, to our knowledge, to consider the suffix tree reverse engineering problem *without the relaxation of suffix links availability*.
- Providing a detection procedure for general alphabet periodic strings suffix tree structure. Given an ordered tree T with n leaves, our detection algorithm takes $O(n + |\Sigma|^p)$ -time, where p is the *unknown in advance length* of a period that repeats at least 3 times in a string S having a suffix tree structure identical to T , if such S exists. Therefore, it is a polynomial time algorithm if p is a constant and a linear time algorithm if, in addition, the alphabet has a sub-linear size.¹
- Proving several necessary (but insufficient) conditions for binary alphabet general strings suffix tree detection.

By this we take another step towards understanding suffix trees structure.

Paper Organization. The paper is organized as follows. In Sect. 2 we give some basic needed formal terms. In Sect. 3 we study the general alphabet periodic strings suffix tree detection problem. In Sect. 4 we prove some necessary (but insufficient) conditions for a binary alphabet general string suffix tree detection, which can be validated using a linear time algorithm. We conclude the paper in Sect. 5 with some open problems.

¹ Though Sect. 3 studies periodic string having period length at least 2, which we call *non-trivial periodic strings*, this complexity is also valid for detection of trivial periodic strings (i.e., unary strings), as follows from condition 2 of Theorem 2 in Sect. 4.

2 Preliminaries

In this section we describe some basic terms used in the paper. We begin with a formal definition² of the term *suffix tree*, which is the subject of this study.

Definition 2. Suffix Tree [21,25,26]

The suffix tree for the string S of length n is defined as a rooted tree such that:

1. The tree has exactly n leaves numbered from 1 to n .
2. Except for the root, every internal node has at least two children.
3. Each edge is labelled with a non-empty sub-string of S .
4. No two edges starting out of a node can have string-labels beginning with the same character. The string obtained by concatenating all the string-labels found on the path from the root to leaf i is the suffix $S[i..n]$, $1 \leq i \leq n$.

In order to ensure the existence of such a tree for any string, S is assumed to end with a terminal symbol denoted by $\$$, where $\$ \notin \Sigma$. This ensures that no suffix is a prefix of another, and that there are n leaf nodes, one for each of the $n - 1$ suffixes of S and an additional leaf for the $\$$.

Throughout the paper, we assume that the suffix tree (thus, any given tree to be checked) is ordered, i.e., the labels on edges are sorted by lexicographic order, and that a $\$$ is lexicographically smaller than every other character in Σ .

Notation. We denote by $|S|$ the length of a string S . Let T be a suffix tree of a string S .

- We denote by $T(\sigma_i)$ the sub-tree of T 's root that is reached by following the edge-label starting with $\sigma_i \in \Sigma$, if σ_i appears in S .
- Similarly, let X be any internal node of T and T_X be the sub-tree rooted at X in T , we denote by $T_X(\sigma_i)$ the sub-tree of X that is reached by following the edge-label starting with $\sigma_i \in \Sigma$, if it exists.

Common Terms. We use some common definitions and terms, as follows.

- Trees terms, such as nodes **level** (starting from 0 for the root), **depth** or **degree**, refer to the unlabelled-edges suffix tree structure.
- **Implicit** nodes in a suffix tree T refer to any non-branching nodes that do not exist in T but can be added between the nodes of T when edge-labels longer than a single character are split to form shorter edge labels.
- **The path label** of a non-root internal node X in T refers to the concatenation of edge-labels on the path from the root to X .

A well-known term related to the suffix tree data structure is that of a *suffix link*, given in Definition 3. Suffix links are a key feature for some classical linear-time construction algorithms [21,25,26], although Farach's algorithm for suffix tree construction [10] does not make use of them. Suffix links are also used in some algorithms running on the tree.

² Assuming the basic terms: rooted trees, internal tree nodes and tree leaves are known.

Definition 3. Suffix Links

Given a suffix tree T , let s be a path label of a non-root internal node X , and let s' be the string s truncated by its first character. Let X' be a non-root internal node with the path label s' . Then, the suffix tree edge from the node X to the node X' is called a suffix link.

In Sect. 3 we deal with periodic strings, formally defined next.

Definition 4. Periodic and Primitive Strings

Let S be a string of length n . S is called periodic if $S = P^i \text{pref}(P)$, where $i \in \mathbb{N}$, $i \geq 2$, P is a substring of S such that $|P| \leq n/2$, P^i is the concatenation of P to itself i times, and $\text{pref}(P)$ is a prefix of P . The shortest such substring P is called the period of S .

If S is not periodic it is called aperiodic or primitive.

A periodic string having a period length at least 2 is called a non-trivial periodic string.

Note that by Definition 4, the period P of a periodic string S is uniquely defined. In addition, such a period P is a primitive string.

Notation. Let S be a periodic string having a period $P = P[1 \dots p]$ of length $p \geq 2$. Denote by $P^{(i)} = P[i..p]P[1..i-1]$, the i -th rotation of P , where $1 \leq i \leq p$.

Definition 5 describes a binary tree structure characterization that we use.

Definition 5. Tree/Sub-Tree Alignment

Let T be a full binary tree, i.e. each node is either a leaf or has exactly two children. T is called left aligned (respectively, right aligned) if every right child (respectively, left child) of an internal node in T is a leaf. We call this the left-alignment condition (respectively, right-alignment condition).

A tree T with only a root and two leaves is called trivially right-aligned (alternatively, trivially left-aligned).

A tree T is not aligned if it is neither left- nor right-aligned.

A sub-tree of T with a root t is called left aligned sub-tree (respectively, right aligned) if the left-alignment condition (respectively, the right-alignment condition) holds for the sub-tree of T rooted at t . The sub-tree of T rooted at t is called non-aligned sub-tree if it is neither a left nor a right aligned sub-tree.

Note that the left and right child of an inner node in a full binary tree representing a suffix tree, correspond to following the edge with a smaller or larger character by lexicographic order, respectively..

For the statement of condition 2 of Theorem 2 (stating necessary conditions for binary string suffix tree detection) in Sect. 4, we need the term *alignment inversion* and the classification of its types given next in Definition 6.

Definition 6. Alignment Inversion

Let T be a binary tree with root node t . Denote by t_ℓ the left child of t and by t_r the right child of t . We say that t has an alignment inversion if one of the following conditions holds:

1. If t_ℓ has 2 children, t_r is a leaf, whereas, the left child of t_ℓ , denoted by t_{ℓ_ℓ} is a leaf, and its right child, denoted by t_{ℓ_r} , has 2 children. We call this a left-right inversion.
2. If t_ℓ is a leaf, t_r has 2 children, whereas, the left child of t_r , denoted by t_{r_ℓ} , has 2 children, and its right child, denoted by t_{r_r} , is a leaf. We call this a right-left inversion.
3. If both t_ℓ and t_r have 2 children. We call this a balanced inversion.

Remark. In Sect. 4 we assume that $|\Sigma| = 2$, without loss of generality, $\Sigma = \{a, b\}$.

3 Periodic Strings Suffix Tree Detection

In this section we study the structure of any ordered tree representing a suffix tree of a non-trivial periodic string, i.e., having period length at least 2.³ We begin by discussing limitations of periodic strings suffix tree detection. Note that in this section, we refer to a string S ending with a \$ as periodic (resp. a-periodic), if it is periodic (resp. a-periodic) when the \$ is omitted.

Detection Limitations. A major limitation for periodic strings suffix tree structure characterization is non-uniqueness, i.e., there exist trees structures that *match both a periodic and a non-periodic string*. Consider the periodic string $babbbabbbba\$$ and the non-periodic string $bbbabbbabbbaba\$$, which have identical suffix trees structure, but the “period” is violated in the incomplete “occurrence” at the end of the non-periodic string. Another different example is the periodic string $babbbabbbabb\$$ and the non-periodic string $babbbabbbabb\$$, in which the last “occurrence” of the period has the correct quantity of each character but their order is different, however, both strings have identical suffix tree structure. We conclude with the following periodic string: $bbabbbabbbabbbabb\$$ and the non-periodic string $bbbbbbabbbabbbabb\$$, which has one mismatch error with the periodic string. They don’t have identical suffix tree structure, however, the suffix tree structure of the second non-periodic string is identical to the suffix tree structure of the following periodic string: $bbbbabbbabbbabb\$$. Thus, violating the periodicity in a complete period occurrence, does not guarantee the in-existence of a periodic string suffix tree that is isomorphic to the resulting aperiodic string suffix tree structure. Therefore, our study is directed to differentiate between ordered trees such that *there exists* a periodic string with identical suffix tree structure and ordered trees where *there is no periodic string suffix tree* with identical structure.

Another important limitation for periodic strings suffix tree structure characterization is having a sufficient number of period repetitions. By periodicity definition (see Definition 4) there should be at least two period repetitions in a periodic string. However, we show that given an ordered tree, determining

³ Note that the structure of a unary string, which is a trivial periodic string with period length 1, is characterized by Theorem 2.2 in Sect. 4.

if it is isomorphic to a periodic string suffix tree can be done for strings with at least *three* period repetitions. Having a sufficient number of repetitions is also a limitation in the *period recovery problem* [1] as well as the *cover recovery problem* [2].

Lemmas 1, 2 and 3 below specify necessary conditions on non-trivial periodic strings suffix trees structure. Omitted proofs will appear in the full version of this paper.

Lemma 1. *Let T be an ordered suffix tree of a periodic string S of length n having a period P of length $p \geq 2$. Then, every internal node at depth at least $p - 1$ has out-degree 2. Moreover, every internal node at depth at least $p - 1$ is a root of a (maybe trivially) right-aligned sub-tree.*

Proof. By the definition of a period, P is primitive, thus every $P^{(i)}$ and $P^{(j)}$, where $i \neq j \pmod p$, have at least one mismatch. Therefore, every two suffixes S_i and S_j , where $i \neq j \pmod p$, have at least one mismatch in their first p characters. Thus, at depth higher than $p - 1$ an internal node can have out-degree more than 2 if at least 3 suffixes share a prefix, where the mismatches pairs are different.

Since there are at most p mismatches in the first p characters of every S_i and S_j , where $i \neq j \pmod p$, then from depth $p - 1$ all suffixes that are still in the same sub-tree have equivalent length modulo p . The only mismatch that creates a new internal node in this set of suffixes is when a shorter suffix ends with a \$, while all the other suffixes are longer. Since all these suffixes have equivalent length modulo p , in all the longer suffixes this character is equal to $S[n - p]$.

We prove that the resulting sub-tree at depth $p - 1$ is a right-aligned sub-tree, from which, in particular, we get that every internal node at such a sub-tree has out-degree 2. The order between pairs of suffixes S_i and S_j , where $i = j \pmod p$, is determined by the comparison of \$ to the character $S[n - p]$. Recall that \$ is smaller and thus forms a left leaf. By periodicity, $S[n - p]$ repeats every p positions, therefore, the repeated comparison forms a right-aligned sub-tree.

Lemma 2. *Let T be an ordered suffix tree of a periodic string S of length n having a period P of length $p \geq 2$. Then, if $\lfloor (n - 1)/p \rfloor > 2$ (i.e., the period appears at least 3 times), there are exactly p sub-trees with depth at least $\lfloor (n - 1)/p \rfloor - 2$ that are all (maybe trivially) right-aligned. Moreover, for every $\sigma \in \Sigma$, the number of such sub-trees in $T(\sigma)$ is the number of occurrences of σ in P .*

Proof. From Lemma 1, it follows that after a prefix of length at most p until the closest appearance of the last p characters of S (excluding the \$), each set of suffixes that have equivalent length modulo p are in the same right-aligned sub-tree.

Each sub-tree corresponds to a different set of suffixes S_i having equivalent length modulo p . Since $\lfloor \frac{n-1}{p} \rfloor > 2$, each such set of suffixes S_i having equivalent length modulo p is not empty and has at least 2 suffixes. Thus, there must be exactly p such right-aligned sub-trees.

Moreover, the number of suffixes in each of these sets is at least: $\lfloor \frac{n-1}{p} \rfloor - 1$. In each level one such suffix ends as a \$-labelled left leaf. Therefore, the number

of internal nodes in the each formed right-aligned sub-tree is at least $\lfloor \frac{n-1}{p} \rfloor - 1$. Thus, the depth of the formed right-aligned sub-tree is at least $\lfloor (n-1)/p \rfloor - 2$.

Since the period length is p , every S_i which belongs to the same sub-tree starts with the same character. That is, $S_i[1] = S_{i+p}[1] = S_{i+2p}[1] = \dots$, for any $1 \leq i \leq p$. Since different length-equivalence classes modulo p are formed by mismatches in the first p characters of S , the number of such right-aligned sub-trees in $T(\sigma)$, for every character σ , is the number of occurrences of σ in P .

Lemma 3. *Let T be an ordered suffix tree of a periodic string S of length n having a period P of length $p \geq 2$. Then, the label of every edge:*

1. *to an inner node of T : has length at most p .*
2. *to a left leaf of T : has length 1 and its label is \$.*
3. *to a right leaf of T : has length $p + 1$ and its label is the suffix $S[n - p \dots n]$.*

Lemmas 1, 2 (and 3) are not sufficient to reject any tree structure where no periodic string has an identical suffix tree structure. However, they can be used to reject the suffix tree of the following **almost** periodic string (with one deletion error): $bbbabbbbbbabbbabb\$$. Lemma 4 below rejects its structure as matching any periodic string suffix tree.

Lemma 4. *Let T be an ordered suffix tree of a string of length n , then if the following hold:*

1. *There exists p , such that $2 \leq p < n$, $\lfloor (n-1)/p \rfloor > 2$ and T has p right-aligned sub-trees.*
2. *There exists a sub-tree at depth $p - 1$ that is not right-aligned.*

Then, there is no periodic string with a suffix tree structure identical to T .

Lemmas 1, 2, 3 and 4 are still not sufficient to reject any tree structure where no periodic string has an identical suffix tree structure. For example, they hold for the suffix tree of the following non-periodic string $babbabbbbbbabbbabb\$$, yet there is no periodic string having an identical suffix tree structure. Nevertheless, Lemma 5 below rejects its structure as matching any periodic string suffix tree.

Lemma 5. *Let T be an ordered suffix tree of a string of length n , then if the following hold:*

1. *There exists p , such that $2 \leq p < n$, $\lfloor (n-1)/p \rfloor > 2$ and T has p right-aligned sub-trees.*
2. *The difference in depths of two of these p right-aligned sub-trees is strictly greater than 1.*

Then, there is no periodic string with a suffix tree structure identical to T .

Proof. If there exists p , such that $2 \leq p < n$, $\lfloor (n-1)/p \rfloor > 2$ and T has p right-aligned sub-trees, then by Lemma 2, the period of a periodic string with a suffix tree T (if such a string exists) is of length p . We next show that the difference in the depth of any two of these right-aligned sub-trees is at most 1.

Each right-aligned sub-tree corresponds to a different set of suffixes having equivalent length modulo p (see Lemma 2's proof). Moreover, the number of suffixes in each of these sets is at least: $\lfloor \frac{n-1}{p} \rfloor - 1$. Thus, their depth is at least $\lfloor (n-1)/p \rfloor - 2$. Also, note that the number of suffixes with equivalent length modulo p in each such set cannot exceed $\lceil (n-1)/p \rceil$. It follows that the difference in the depth of any two such sub-trees is at most 2.

Assume that there exists a non-empty set of $\lfloor (n-1)/p \rfloor - 1$ suffixes having equivalent length modulo p and let S_i be the longest suffix in this set. The length of S_i is, therefore, $r + p(\lfloor (n-1)/p \rfloor - 2) + 1$ (including the \$), where r is the length of the path label of S_i 's right-aligned sub-tree root. We claim that in such a case there is no other set with $\lceil (n-1)/p \rceil$ suffixes having equivalent length modulo p . Note that proving this claim concludes the proof of the lemma.

Every suffix S_j , where $i \neq j \pmod p$, has a mismatch with S_i which must be encountered after at most $p-1$ characters. Note that, $n-1 - (p-1) \geq p\lfloor (n-1)/p \rfloor - p + 1 = p(\lfloor (n-1)/p \rfloor - 1) + 1 = p(\lfloor (n-1)/p \rfloor - 2) + p + 1$. This means that the shortest suffix with equivalent length to S_i modulo p has the same prefix of length p as S_i and, therefore, its length is at least $p+1$ (including the \$). Thus, $r \geq p$, since this shortest suffix ends with a \$ as a left leaf of the root of S_i 's right-aligned sub-tree. Moreover, the label of the edge leading to the root of S_i 's right-aligned sub-tree is of length p . Therefore, $r \leq p-1 + p < 2p$.

Now, every other set of suffixes with equivalent length modulo p that has more suffixes than $\lfloor \frac{n-1}{p} \rfloor - 1$, must have a longest suffix with length strictly greater than that of S_i . Since $r < 2p$, such a suffix must begin within the first $2p-1$ characters of S . Therefore, it can have at most one more suffix than the number of suffixes in set of S_i . This concludes the proof.

Theorem 1. *Let T be an ordered tree with n leaves. Then, there exists a periodic string S of length n having a period P of length $p \geq 2$ that repeats at least 3 times in S s.t. T is identical to the structure of S 's suffix tree, if and only if:*

1. *Every internal node at depth at least $p-1 \geq 1$ has out-degree 2.*
2. *There are exactly p sub-trees with depth at least $\lfloor (n-1)/p \rfloor - 2 \geq 1$ that are all (maybe trivially) right-aligned. Moreover, for every $\sigma \in \Sigma$, the number of such sub-trees in $T(\sigma)$ is the number of occurrences of σ in P .*
3. *Every sub-tree at depth $p-1$ is right-aligned.*
4. *The depths difference between every two of the p right-aligned sub-trees of T is at most 1.*
5. *Let T' be the tree T in which the p right-aligned sub-trees that T has by condition 2 are trimmed at their sub-trees roots. Then, there exists an a-periodic string \hat{P} of length x , such that $p+2 \leq x \leq 2p$, $\hat{P}[i] = \hat{P}[p+i]$, for $1 \leq i \leq x-p-1$, $\hat{P}[x] = \$$ and the suffix tree of \hat{P} is identical to T' .*

Proof. Assume that there exists a periodic string S of length n having a period P of length $p \geq 2$ that repeats at least 3 times in S such that T is identical to the structure of S 's suffix tree. Then, Lemmas 1–5 ensure that conditions 1–4 hold. In addition, the set of at most $2p$ shortest suffixes of S determine the structure of T above the p right-aligned sub-trees that T has by condition 2. This is because

at least one and at most p mismatches occur between the suffix of length $p + 2$ to every other shorter suffix. The mismatches of the $p + 2$ shortest suffixes result in inner nodes above the p right-aligned sub-trees, since these are suffixes that do not have equivalent length modulo p . Suffixes with length $p + 2 < x \leq 2p$ may still create an edge to a root of the p right-aligned sub-trees. Any longer suffix has equivalent length modulo p to one of the suffixes of length at most $p + 1$. Therefore, there exists x , $p + 2 \leq x \leq 2p$, such that by taking \hat{P} to be the x -length suffix of S , we get that the suffix tree of \hat{P} is identical to T' . Note that \hat{P} is a-periodic (having length at most $2p$ including the \$).

Assume that conditions 1–5 hold for an ordered tree T with n leaves, we show that there exists a periodic string S of length n having a period P of length $p \geq 2$ that repeats at least 3 times in S such that T is identical to the structure of S 's suffix tree. By condition 5, we know that there exists an a-periodic string \hat{P} of length x , such that $p + 2 \leq x \leq 2p$, $\hat{P}[i] = \hat{P}[p + i]$, for $1 \leq i \leq x - p - 1$, $\hat{P}[x] = \$$ and the suffix tree of \hat{P} is identical to T' , where T' is the tree T in which the p right-aligned sub-trees that T has by condition 2 are trimmed at their sub-trees roots. We inductively construct the periodic string S from end to beginning as follows. We begin by assigning $S[n - x + 1 \dots n] \leftarrow \hat{P}$. Then, for every i , starting from $n - x$ down to 1, we assign $S[i] \leftarrow S[i + p]$. Note that S is periodic with a period of length p . By condition 1, we have that $p \geq 2$ and by condition 2, we have that $\lfloor (n - 1)/p \rfloor > 2$ (thus, the period repeats at least 3 times in S). Let T'' be the suffix tree of S . Then, Lemmas 1–5 ensure that conditions 1–4 hold for T'' . Since the construction of S only added suffixes longer than x which belong to the right-aligned sub-trees, trimming the p right-aligned sub-trees of T'' at their roots gives the tree T' . Therefore, T'' is identical to T . The theorem follows.

Corollary 1. General Alphabet Periodic String Suffix Tree Detection
Given an ordered tree T with n leaves, then there exists an $O(n + |\Sigma|^p)$ -time algorithm to detect if there exists a non-trivial periodic string S of length n over Σ having at least 3 period repetitions, s.t. T is identical to S 's suffix tree structure, where p is the unknown in advance period length of S , if such S exists.

Proof. Given an ordered tree T with n leaves, the algorithm's steps are:

1. Scan T to find the *unique*⁴ number p of right-aligned sub-trees with depth at least $\lfloor (n - 1)/p \rfloor - 2 \geq 1$ and the numbers p_σ of the right-aligned sub-trees in every $T(\sigma)$ sub-tree of T .⁵
2. If $\lfloor (n - 1)/p \rfloor \leq 2$ or $p < 2$, return NO.
3. If two of these p right-aligned sub-trees have depth difference strictly greater than 1, return NO.
4. Scan T to check if every internal node at depth at least $p - 1$ has out-degree 2. If not, return NO.

⁴ p is unique by condition 2 of Theorem 1.

⁵ The characters $\sigma \in \Sigma$ are chosen to the edges of T 's root from left to right according to the order of Σ , where the first edge is \$.

5. If the scan detects a sub-tree at depth $p - 1$ that is not right-aligned, return NO.
6. Let T' be the tree T in which the p right-aligned sub-trees of T are trimmed at their sub-trees roots and let x be the number of leaves in T' .
7. For every a-periodic string \hat{P} of length x , such that $\hat{P}[i] = \hat{P}[p + i]$, for $1 \leq i \leq x - p - 1$, $\hat{P}[x] = \$$, having p_σ occurrences of each $\sigma \in \Sigma$ in the last $p + 1$ characters:
 - (a) Construct the suffix tree $T^{\hat{P}}$ of the string \hat{P} .
 - (b) If $T^{\hat{P}}$ is identical to T' , return YES.
8. Return NO.

The algorithm correctness: Step 1 identifies the only possible candidate period length p according to condition 2 of Theorem 1. Then, step 2 rejects T if this candidate is not long enough or for this candidate there are not enough repetitions, as we only detect suffix trees of non-trivial periodic strings with at least three period repetitions. Step 3 rejects T according to condition 4 of Theorem 1. Step 4 rejects T according to condition 1 of Theorem 1. This step is performed after step 3 to prevent another unnecessary scan of T if the information of the first three steps is enough to reject T . Step 5 rejects T according to condition 3 of Theorem 1. Steps 6–8 reject or accept T according to condition 5 of Theorem 1.

The algorithm time complexity is $O(n + |\Sigma|^p)$, as steps 1– take time linear in the size of T , which is $O(n)$, step 7 takes $O(|\Sigma|^p + p)$ time (recall that $p + 2 \leq x \leq 2p$ by condition 5 of Theorem 1) and step 8 takes $O(1)$ time.

4 Necessary Conditions on a Binary String Suffix Tree

In this section we describe some necessary conditions on any ordered tree representing a binary string suffix tree. For completeness, we state Observation 1, which follows from Definition 2 and the binary string ordered tree assumption.

Observation 1 *Let T be an ordered suffix tree of a binary string of length n . Then,*

1. *Every internal node of T has out-degree at least 2 and at most 3.*
2. *The leftmost child of the root of T has out-degree 0 (no children). This child corresponds to the \$ character in the string represented by T .*
3. *The edge to the leftmost child among three children of any node in T is labelled with \$.*

Theorem 2 below lists several necessary conditions for a tree T to be a suffix tree of a binary string.

Theorem 2. Necessary Conditions

Let T be an ordered suffix tree of a binary string of length n . Then,

1. *The maximum out-degree (number of children) of nodes in the same depth is monotonically non increasing with the node depth. I.e., if there exists an internal node with out-degree 3, then all the above levels must have at least one node with out-degree 3.*
2. *Let $d \geq 0$ be the highest level at which the maximum out-degree of the nodes is 2. Then the following hold:*
 - *If $d = 0$, the tree is right-aligned.*
 - *If $d = 1$, the root of $T(a)$ cannot have left-right inversion, however, it can have right-left or balanced inversion. On the other hand, the root of $T(b)$ can either be aligned or have left-right, right-left or balanced inversion.*
3. *The maximum number of leaves in $T_X(a)$, for an internal node X , is monotonically non-increasing with the depth of X .*
4. *The maximum number of leaves in $T_X(b)$, for an internal node X , is monotonically non increasing with the depth of X .*
5. *If all the possible strings of length 2: aa , ab , ba and bb , appear in the string represented by T , then the root of T has only one child with out-degree 3.*

The proof of Theorem 2 is deferred to the full version.

We conclude this section by referring to the algorithmic task of verifying if the conditions specified in (Observation 1 and) Theorem 2 hold for a given ordered tree T , assuming a binary string. Theorem 3 below can be achieved simply by visiting the tree.

Theorem 3. Verification Algorithm

Given an ordered tree T , there exists an $O(n)$ time algorithm to verify if each of the conditions of (Observation 1 and) Theorem 2 holds, assuming a binary string.

Remark. Theorem 2's conditions are insufficient.

5 Conclusion and Open Problems

In this paper we made another step towards understanding the structure of suffix trees by studying the suffix tree detection problem in which the relaxation of having the suffix links as input is removed. Some interesting open problems are:

- Providing a full characterization theorem describing both necessary and sufficient conditions for an ordered tree to be a suffix tree of some string.
- Improving the time complexity of our general alphabet periodic string suffix tree detection algorithm.

We believe that these problems should be further addressed due to the importance of suffix trees and their fundamental role.

References

1. Amir, A., Eisenberg, E., Levy, A., Porat, E., Shapira, N.: Cycle detection and correction. *ACM Trans. Algorithms* **9**(1), 1–20 (2012). <https://doi.org/10.1145/2390176.2390189>
2. Amir, A., Levy, A., Lewenstein, M., Lubin, R., Porat, B.: Can we recover the cover? *Algorithmica* **81**(7), 2857–2875 (2019). <https://doi.org/10.1007/s00453-019-00559-8>
3. Apostolico, A.: The myriad virtues of subword trees. In: Apostolico, A., Galil, Z. (eds.) *Combinatorial Algorithms on Words*. NATO ASI Series, vol. 12, pp. 85–96. Springer, Heidelberg (1985). https://doi.org/10.1007/978-3-642-82456-2_6
4. Bannai, H., Inenaga, S., Shinohara, A., Takeda, M.: Inferring strings from graphs and arrays. In: Rován, B., Vojtáš, P. (eds.) *MFCS 2003*. LNCS, vol. 2747, pp. 208–217. Springer, Heidelberg (2003). https://doi.org/10.1007/978-3-540-45138-9_15
5. Breslauer, D., Italiano, G.F.: On suffix extensions in suffix trees. *Theoret. Comput. Sci.* **457**, 27–34 (2012)
6. Cazaux, B., Rivals, E.: Reverse engineering of compact suffix trees and links: a novel algorithm. *J. Discrete Algorithms* **28**, 9–22 (2014)
7. Clément, J., Crochemore, M., Rindone, G.: Reverse engineering prefix tables. In: *Proceedings of the 26th International Symposium on Theoretical Aspects of Computer Science STACS*. LIPIcs, vol. 3, pp. 289–300. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany (2009)
8. Crochemore, M., Iliopoulos, C.S., Pissis, S.P., Tischler, G.: Cover array string reconstruction. In: Amir, A., Parida, L. (eds.) *CPM 2010*. LNCS, vol. 6129, pp. 251–259. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-13509-5_23
9. Duval, J., Lecroq, T., Lefebvre, A.: Efficient validation and construction of border arrays and validation of string matching automata. *RAIRO Theor. Inform. Appl.* **43**(2), 281–297 (2009)
10. Farach, M.: Optimal suffix tree construction with large alphabets. In: *Proceedings of the 38th IEEE Symposium on Foundations of Computer Science*, pp. 137–143 (1997)
11. Franek, F., et al.: Verifying a border array in linear time. *J. Comb. Math. Comb. Comput.* **42**, 223–236 (2000)
12. Gawrychowski, P., Jez, A., Jez, L.: Validating the Knuth-Morris-Pratt failure function, fast and online. *Theory Comput. Syst.* **54**(2), 337–372 (2014)
13. Gawrychowski, P., Kociumaka, T., Radoszewski, J., Rytter, W., Walen, T.: Universal reconstruction of a string. *Theor. Comput. Sci.* **812**, 174–186 (2020)
14. Gelle, K., Iván, S.: Recognizing union-find trees is NP-complete, even without rank info. *Int. J. Found. Comput. Sci.* **30**(6–7), 1029–1045 (2019)
15. Gusfield, D.: *Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology*. Cambridge University Press, Cambridge (1997)
16. He, M., Munro, J.I., Rao, S.S.: A categorization theorem on suffix arrays with applications to space efficient text indexes. In: *SODA*, vol. 5, pp. 23–32. Citeseer (2005)
17. Tomohiro, I., Inenaga, S., Bannai, H., Takeda, M.: Verifying and enumerating parameterized border arrays. *Theor. Comput. Sci.* **412**(50), 6959–6981 (2011)
18. Tomohiro, I., Inenaga, S., Bannai, H., Takeda, M.: Inferring strings from suffix trees and links on a binary alphabet. *Discrete Appl. Math.* **163**, 316–325 (2014)

19. Kärkkäinen, J., Piatkowski, M., Puglisi, S.J.: String inference from longest-common-prefix array. In: Proceedings of the 44th International Colloquium on Automata, Languages, and Programming, ICALP. LIPIcs, vol. 80, pp. 62:1–62:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2017)
20. Kucherov, G., Tóthmérés, L., Vialette, S.: On the combinatorics of suffix arrays. *Inf. Process. Lett.* **113**(22), 915–920 (2013)
21. McCreight, E.M.: A space-economical suffix tree construction algorithm. *J. ACM* **23**, 262–272 (1976)
22. Nakashima, Y., Okabe, T., Tomohiro, I., Inenaga, S., Bannai, H., Takeda, M.: Inferring strings from Lyndon factorization. *Theor. Comput. Sci.* **689**, 147–156 (2017)
23. Schürmann, K.B., Stoye, J.: Counting suffix arrays and strings. *Theoret. Comput. Sci.* **395**(2–3), 220–234 (2008)
24. Starikovskaya, T., Vildhøj, H.W.: A suffix tree or not a suffix tree? *J. Discrete Algorithms* **32**, 14–23 (2015)
25. Ukkonen, E.: On-line construction of suffix trees. *Algorithmica* **14**, 249–260 (1995)
26. Weiner, P.: Linear pattern matching algorithm. In: Proceedings of the 14 IEEE Symposium on Switching and Automata Theory, pp. 1–11 (1973)