Inferring Strings from Suffix Trees and Links on a Binary Alphabet

Tomohiro I, Shunsuke Inenaga, Hideo Bannai, and Masayuki Takeda

Department of Informatics, Kyushu University
744 Motooka, Nishi-ku, Fukuoka 819–0395, Japan
{tomohiro.i,inenaga,bannai,takeda}@inf.kyushu-u.ac.jp

Abstract. A suffix tree, which provides us with a linear space full-text index of a given string, is a fundamental data structure for string processing and information retrieval. In this paper we consider the reverse engineering problem on suffix trees: Given an unlabeled ordered rooted tree T accompanied with a node-to-node transition function f, infer a string whose suffix tree and its suffix links for inner nodes are isomorphic to T and f, respectively. By introducing new characterizations of suffix trees, we show that the reverse engineering problem on suffix trees on a binary alphabet can be solved in linear time in the input size.

1 Introduction

1.1 Suffix Trees

Suffix trees, one of the most well known and widely-used text indexing structures, have played a central role in combinatorial pattern matching and its applications. A multitude of important problems can efficiently be solved using suffix trees [1,10].

A suffix tree of a string w is a compacted trie which represents all the suffixes of w. Each edge of the suffix tree is labeled with a substring y of w, and the edge string y is represented by a pair (i, j) of positions such that the substring of w that begins at position i and ends at position j is identical to y. In this way the suffix tree can be represented with linear space in the length of w. Linear-time suffix tree construction algorithms proposed in [19,15,18] utilize auxiliary edges called *suffix links*. There exists a suffix link from node v to node u if the substring represented by u is identical to the string that is obtained by removing the first character of the substring represented by v.

1.2 Our Contribution

We consider the reverse engineering problem on suffix trees, i.e., given an ordered rooted tree T with its edges *unlabeled*, determine whether there exists a string w such that the edge-unlabeled suffix tree of w is isomorphic to T. If one exists, then output such a string. We emphasize that this problem is very challenging, intuitively, due to the following reasons:

- The length of each edge string is not given.
- The mapping from strings to edge-unlabeled suffix trees is not injective.

As a first step towards solving the problem, we restrict the alphabet to a binary one. Also, we assume that suffix links of inner nodes are given as input. We show that, with these conditions, we can solve the reverse engineering problem on suffix trees in linear time in the size of the input tree T. We remark that if suffix links of leaves are also given, then the problem can be easily solved in linear time. However, it is much more difficult to reverse engineer a string only from suffix links of inner nodes.

Tomohiro I, Shunsuke Inenaga, Hideo Bannai, Masayuki Takeda: Inferring Strings from Suffix Trees and Links on a Binary Alphabet, pp. 121–131.
Proceedings of PSC 2011, Jan Holub and Jan Žďárek (Eds.), ISBN 978-80-01-04870-2

1.3 Related Work

Inferring a string from other string data structures has been widely studied. An algorithm to find a string having a given border array was presented in [8], which runs in linear time for an unbounded alphabet. A simpler linear-time solution for the same problem for a bounded alphabet was shown in [5]. Linear-time and $O(n^{1.5})$-time inferring algorithms for parameterized versions of border arrays, on a binary alphabet and an unbounded alphabet, respectively, were proposed [11,13]. Linear-time inferring algorithms for suffix arrays [7,2], KMP failure tables [6,9], prefix tables [3], cover arrays [4], palindromic structures [12], directed acyclic word graphs [2] and directed acyclic subsequence graphs [2] have been proposed, which provide us with further insight concerning the data structures. Also, it was recently revealed that the time complexity of reverse problem of runs depends on the alphabet size [14].

Counting and enumerating some of the above-mentioned data structures have also been studied in the literature [16,17,11,13,12].

2 Preliminaries

Let Σ be a finite *alphabet*. An element of Σ^* is called a *string*. The length of a string w is denoted by $|w|$. The empty string ε is a string of length 0, that is, $|\varepsilon| = 0$. Let $\Sigma^+ = \Sigma^* \setminus \{\varepsilon\}$. For a string $w = xyz$, x, y and z are called a *prefix*, *substring*, and *suffix* of w, respectively. The i-th character of a string w is denoted by $w[i]$ for $1 \leq i \leq |w|$, and the substring of a string w that begins at position i and ends at position j is denoted by $w[i:j]$ for $1 \leq i \leq j \leq |w|$. For convenience, let $w[i:j] = \varepsilon$ if $j < i$. For any characters $a, b \in \Sigma$, we write as $a \prec b$ if a is lexicographically smaller than b.

Let $T = (V_T, E_T)$ be an ordered rooted tree. The root node of T is denoted by $\bot_T$. V_T^{in} and V_T^{leaf}, respectively, denote the set of the inner nodes and the set of the leaf nodes of T. For any $v \in V_T$, let $V_T(v) = V_{T'}$, $V_T^{in}(v) = V_{T'}^{in}$ and $V_T^{leaf}(v) = V_{T'}^{leaf}$, where T' is the subtree of T rooted at v. For any $v \in V_T$, the set of children of v, the i-th child of v and the parent of v are denoted by $\mathsf{children}(v)$, $\mathsf{child}_i(v)$ and $\mathsf{par}(v)$, respectively.

The *suffix tree* of a string w, denoted by $ST(w)$, is a compacted trie which represents all the suffixes of w. Let us assume that w ends with a terminal symbol $\$ \notin \Sigma$, where $\$$ is lexicographically smaller than any character in Σ, so that $ST(w)$ has exactly $|w|$ leaves. Every edge $e \in E_{ST(w)}$ is labeled with a substring of w. We call it the *edge string* of e. For any $v \in V_{ST(w)}^{in}$, all edge strings coming out from v must begin with distinct characters, and the children of v are sorted in lexicographic order of edge strings, namely, for any $1 \leq i < |\mathsf{children}(v)|$, the first symbol of the edge string for $(v, \mathsf{child}_i(v))$ must be lexicographically smaller than that for $(v, \mathsf{child}_{i+1}(v))$. Every node $v \in V_{ST(w)}$ corresponds to a string obtained by concatenating edge strings on the path from $\bot_{ST(w)}$ to v.

The *suffix link* $sl_w : V_{ST(w)} \to (V_{ST(w)} \cup \{\top\})$ of $ST(w)$ is a function such that for any $v \in (V_{ST(w)} - \{\bot_{ST(w)}\})$ that corresponds to a string x, $sl_w(v) = u$ where u is the node that corresponds to $x[2:|x|]$. For the root node $\bot_{ST(w)}$, let $sl_w(\bot_{ST(w)}) = \top$, where $\top$ is an auxiliary node.

Suffix tree $ST(w)$ for string $w = \mathtt{ababaaa\$}$ is shown in Figure 1. An auxiliary node $\top$ is abbreviated in the figure.

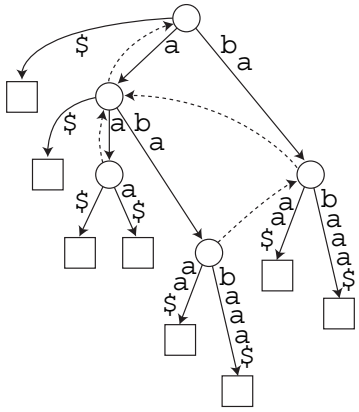

Figure 1. Suffix tree $ST(w)$ and for string $w =$ `abababaa`\$. Suffix links of inner nodes are shown by dotted arrows.

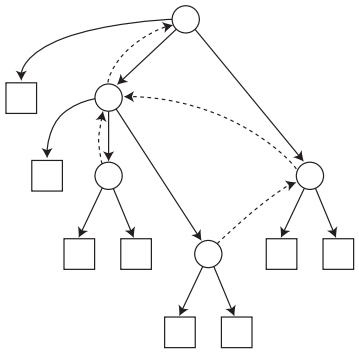

Figure 2. A valid input of Problem 1.

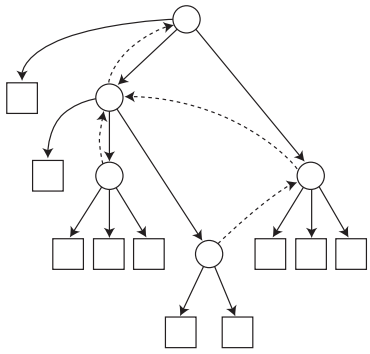

Figure 3. An invalid input of Problem 1.

In this paper, we tackle the following problem:

Problem 1 (Reverse Engineering Problem on Suffix Trees).

Input: An ordered pair (T, f) of an unlabeled ordered rooted tree T and a function $f : V_T^{in} \to (V_T^{in} \cup \{\top\})$.

Output: A string w for which the unlabeled graph induced from $ST(w)$ and sl_w is isomorphic to the graph induced from T and f, if such exists.

Here we say that a string w *realizes* (T, f) if w is a solution to Problem 1 w.r.t. (T, f). If there exists a string which realizes (T, f), then (T, f) is said to be a valid input.

Figure 2 illustrates a valid input of Problem 1; `aababaa`\$, `aababaa`\$, `abaaaba`\$ and `abababaa`\$ realize the tree. On the other hand, Figure 3 illustrates an invalid input of Problem 1; no strings realize the tree. In the figures, the tree T and the function f of an input (T, f) are depicted by solid arrows and dotted arrows, respectively. Also, the inner nodes and the leaf nodes of T are described by the circles and the squares, respectively.

3 Algorithm

3.1 Characterization of Suffix Trees

In this subsection, we give new characterizations of suffix trees and show how to construct a string whose suffix tree and its suffix links for inner nodes are isomorphic

to a given (T, f). Let us remark that the facts to be presented in this subsection apply also to alphabets of an arbitrary size, not only to binary ones.

Proposition 2. *For any string w, $|\mathsf{children}(\bot_{ST(w)})|$ represents the number of distinct characters occurring in w.*

Since a terminal symbol \$ is the lexicographically smallest and occurs exactly once in w, the following proposition holds.

Proposition 3. *For any string w, $\mathsf{child}_1(\bot_{ST(w)})$ is a leaf node of $ST(w)$.*

The next proposition follows from the definition of suffix links.

Proposition 4. *For any string w and $v_0 \in V^{in}_{ST(w)}$, there exists a sequence $v_1, v_2, \ldots, v_k$ of nodes such that $v_k = \top$ and $v_i = sl_w(v_{i-1})$ for any $1 \leq i \leq k$.*

The above proposition says that there exists a path of suffix links from any inner node v_0 to the auxiliary node $\top$.

Proposition 5. *For any string w and $u, v \in V^{in}_{ST(w)}$, if $sl_w(u) = sl_w(v)$ and the longest common ancestor between u and v is not $\bot_{ST(w)}$, then $u = v$.*

Proof. Let x and x' be the strings corresponding to u and v, respectively. $sl_w(u) = sl_w(v)$ implies that $|x| = |x'|$ and $x[2 : |x|] = x'[2 : |x'|]$. Since the longest common ancestor between u and v is not $\bot_{ST(w)}$, $x[1] = x'[1]$. Hence $x = x'$, i.e., u and v are identical. □

The next corollary follows from the above propositions.

Corollary 6. *For any string w and $v \in V^{in}_{ST(w)}$, $|\{u \in V^{in}_{ST(w)} \mid sl_w(u) = v\}| < |\mathsf{children}(\bot_{ST(w)})|$.*

Let (T, f) be an input of Problem 1. Since it can be checked in linear time whether (T, f) satisfies the conditions of Propositions 3, 4 and 5, in what follows we assume that (T, f) satisfies those conditions. Also, since every string terminates with an endmarker \$, we assume that every inner node of T has at least two children. In addition, we assume $\mathsf{par}(\bot_T) = \top, \mathsf{children}(\top) = \{\bot_T\}$ and $f(\bot_T) = \top$.

For any $v \in V_T$, we define $sldepth(v)$ by the number of links from v to the root node, namely,

$$sldepth(v) = \begin{cases} 0 & \text{if } v = \bot_T, \\ sldepth(f(v)) + 1 & \text{if } v \in (V^{in}_T - \bot_T), \\ sldepth(\mathsf{par}(v)) + 1 & \text{if } v \in V^{leaf}_T. \end{cases}$$

Lemma 7. *Let (T, f) be an input of Problem 1. If $f(v)$ is a descendant of $f(\mathsf{par}(v))$ for any $v \in (V^{in}_T - \{\bot_T\})$, then $sldepth(\mathsf{par}(v)) < sldepth(v)$ holds for any $v \in V_T$.*

Proof. When v is a leaf node, by definition $sldepth(v) = sldepth(\mathsf{par}(v)) + 1$. Then we prove that for any $u \in V^{in}_T$ and $v \in (V^{in}_T(u) - \{u\})$, $sldepth(u) < sldepth(v)$, by induction on the value of $sldepth(v)$. As a base statement, when $u = \bot_T$, the statement holds due to $sldepth(\bot_T) = 0$. As an induction step, assume that the statement holds for any $u \in V^{in}_T$ with $sldepth(u) < k$, and consider any $u \in V^{in}_T$ with $sldepth(u) = k$. Since $f(v)$ is a descendant of $f(u)$ and $sldepth(f(u)) = k - 1$, $sldepth(f(u)) < sldepth(f(v))$ holds, and hence, $sldepth(u) = sldepth(f(u)) + 1 < sldepth(f(v)) + 1 = sldepth(v)$. Therefore, the lemma holds. □

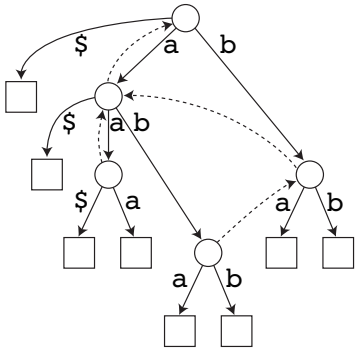

Figure 4. A valid labeling to the input shown in Figure 2.

We consider a labeling function $g : E_T \to (\Sigma \cup \{\$\})$ that associates each edge with a character in $\Sigma \cup \{\$\}$. For ease of description, we let $g((\top, \bot_T))$ denote an arbitrary character in $\Sigma \cup \{\$\}$, i.e., the auxiliary edge $(\top, \bot_T)$ is labeled with all characters in $\Sigma \cup \{\$\}$. Since we intend g to describe the first character of each edge string, let us assume that g satisfies the following preconditions:

1. $g((\bot_T, \mathsf{child}_1(\bot_T))) = \$$,
2. for any $v \in V_T$ and $1 \leq i < |\mathsf{children}(v)|$, $g((v, \mathsf{child}_i(v))) \prec g((v, \mathsf{child}_{i+1}(v)))$,
3. for any $(\mathsf{par}(v), v) \in E_T$, there exists a child u of $f(\mathsf{par}(v))$ s.t. $g((\mathsf{par}(v), v)) = g((f(\mathsf{par}(v)), u))$ and $f(v) \in V(u)$.

If there is no such function g, then (T, f) is invalid. We can check in linear time whether or not there exists a labeling g for which Preconditions 1, 2 and 3 hold, as follows. Firstly, we set $g((\bot_T, \mathsf{child}_i(\bot_T)))$ for any $1 \leq i < |\mathsf{children}(\bot_T)|$ so that they satisfy Preconditions 1 and 2. Secondly, for any $(\mathsf{par}(v), v) \in E_T$ with $v \in V_T^{in}$ and $\mathsf{par}(v) \neq \bot_T$, we determine $g((\mathsf{par}(v), v))$ according to Precondition 3 (note that they are uniquely determined), and check if they satisfy Precondition 2 or not. Finally, for any $(\mathsf{par}(v), v) \in E_T$ with $\mathsf{par}(v) \neq \bot_T$ and $v \in V_T^{leaf}$, we check if $g((\mathsf{par}(v), v))$ satisfies Preconditions 2 and 3 or not.

A labeling g is said to be a *valid labeling* if there exists a string w which realizes (T, f) and the first character of each edge string of $ST(w)$ coincides with the character defined by g.

Figure 4 illustrates one of the valid labelings to the input shown in Figure 2. Note that the first character of every edge string of $ST(\texttt{ababaaa\$})$ shown in Figure 1 coincides with the valid edge label of the tree shown in Figure 4.

For any labeling g and $v \in V_T$, let

$$L_g(v) = |\{u \in V_T^{leaf} \mid f(\mathsf{par}(u)) = \mathsf{par}(v), g((\mathsf{par}(u), u)) = g((\mathsf{par}(v), v))\}| \text{ and}$$
$$D_g(v) = L_g(v) + \sum_{u \in \mathsf{children}(v)} D_g(u) = \sum_{u \in V_T(v)} L_g(u).$$

Figure 5 illustrates the relationship between a node $v \in V_T$ and the leaves related to $L_g(v)$.

To show intuitive meanings of the above notations, let us consider a valid labeling g, and a string w which realizes (T, f) and g. For any $v \in V_T$ and $u \in V_T^{leaf}$ satisfying $f(\mathsf{par}(u)) = \mathsf{par}(v)$ and $g((\mathsf{par}(u), u)) = g((\mathsf{par}(v), v))$, we see that a string ax occurs exactly once in w, where x is the substring corresponding to v and ax is a prefix of the suffix corresponding to u. In addition, for any ancestor v' (corresponding to

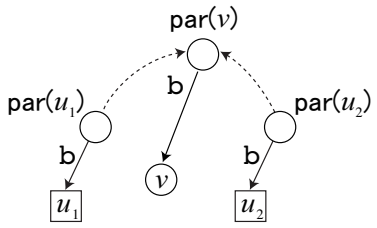

Figure 5. Relationship between a node $v \in V_T$ and the leaves related to $L_g(v)$. In this case, $L_g(v) = 2$.

a substring x') of v, since x' is a prefix of x, one of the occurrences of x' in w is attributed to ax, i.e., to the leaf node u. Note that all the information comes from (T, f) and g, and is independent of a solution w, that is, from the standpoint of the reverse engineering problem, they are constraints to determine w. Hence $D_g(v)$, the sum of the values $L_g(u)$ for all $u \in V_T(v)$, implies the constraints by which v is affected.

The following lemma describes a necessary condition for g to be valid.

Lemma 8. *Let (T, f) be an input of Problem 1 and g be a labeling function. If g is a valid labeling, then $|V_T^{leaf}(v)| \geq D_g(v)$ for any $v \in V_T$.*

Proof. Let w be a string which realizes (T, f) and g. Let $v \in V_T$ and x be a string corresponding to v. Since ax (a is any character in $\Sigma \cup \{\$\}$) occurs at least $D_g(v)$ times in w, $D_g(v)$ is bounded by the number of occurrences of x, that is, $|V_T^{leaf}(v)|$. □

For a labeling function g satisfying the condition of Lemma 8, we introduce a directed multiedge graph, called a *suffix tour graph* w.r.t. g, as follows.

Definition 9 (Suffix Tour Graph). *Let (T, f) be an input of Problem 1 and g be a labeling function such that $|V_T^{leaf}(v)| \geq D_g(v)$ for any $v \in V_T$. The suffix tour graph $STG_g = (V_G, E_G)$ w.r.t. g is defined as follows:*

$$\begin{aligned} V_G &= V_T, \\ E_G &= \{(u, v) \mid u \in V_T^{leaf}, f(\mathsf{par}(u)) = \mathsf{par}(v), g((\mathsf{par}(u), u)) = g((\mathsf{par}(v), v))\} \\ &\quad \cup \{(u, v)^k \mid (u, v) \in E_T, k = |V_T^{leaf}(v)| - D_g(v)\}, \end{aligned}$$

where $(u, v)^k$ is a k-multiedge from u to v.

Figure 7 illustrates the suffix tour graph w.r.t. the labeling shown in Figure 6. In Figure 6, the number in each node v represents the value $|V_T^{leaf}(v)| - D_g(v)$.

Lemma 10. *Let (T, f) be an input of Problem 1 and g be a labeling function. STG_g is an Eulerian graph (possibly disjoint).*

Proof. It suffices to show that the indegree and the outdegree of any $v \in V_T$ are equal. Let v be any node in V_T^{in}. It follows from the definition of E_G that the indegree and outdegree of v are $L_g(v) + |V_T^{leaf}(v)| - D_g(v)$ and $\sum_{u \in \mathsf{children}(v)}(|V_T^{leaf}(u)| - D_g(u))$, respectively. By taking a subtraction between them, we get

$$\begin{aligned} &(L_g(v) + |V_T^{leaf}(v)| - D_g(v)) - \Big(\sum_{u \in \mathsf{children}(v)} (|V_T^{leaf}(u)| - D_g(u))\Big) \\ &= L_g(v) - D_g(v) + \sum_{u \in \mathsf{children}(v)} D_g(u) = 0. \end{aligned}$$

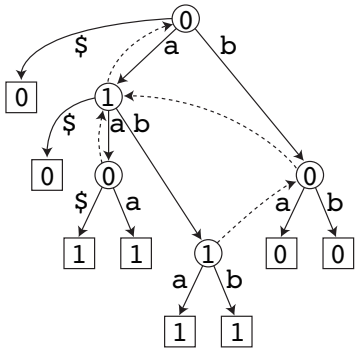

Figure 6. The value $|V_T^{leaf}(v)| - D_g(v)$ of each node v is overlaid on Figure 4.

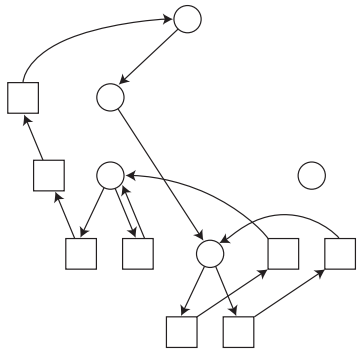

Figure 7. The suffix tour graph w.r.t. the labeling shown in Figure 6.

For any $v \in V_T^{leaf}$, the outdegree of v is 1. Since $D_g(v) = L_g(v)$, in both cases of $D_g(v) = 0$ and $D_g(v) = 1$, the indegree of v is 1. Therefore the lemma holds. □

Lemma 11. *Let (T, f) be an input of Problem 1 and g be a labeling function. The size of $STG_g = (V_G, E_G)$ is $O(|V_T|)$.*

Proof. Since $|V_G| = |V_T|$ and $|\{(u, v) \mid u \in V_T^{leaf}, f(\mathsf{par}(u)) = \mathsf{par}(v), g((\mathsf{par}(u), u)) = g((\mathsf{par}(v), v))\}| = |V_T^{leaf}|$, what we have to do is to estimate the number of edges in $\{(u, v)^k \mid (u, v) \in E_T, k = |V_T^{leaf}(v)| - D_g(v)\}$. It follows from Lemma 7 that $sldepth(u) < sldepth(v)$ for any $(u, v) \in E_G \cap E_T$. Let us call such edges "downward edges". On the other hand, the number of "upward edges", edges $(u, v) \in E_G$ holding $sldepth(u) > sldepth(v)$, is at most $|V_T^{leaf}|$. Since STG_g is an Eulerian graph, the number of "downward edges" is bounded by that of "upward edges", namely, is bounded by $|V_T^{leaf}|$. □

Lemma 12. *Let (T, f) be an input of Problem 1 and g be a labeling function. g is a valid labeling if and only if $STG_g = (V_G, E_G)$ has an Eulerian cycle which contains $\perp_T$ and all leaves of T.*

Proof. **if part:** Let us consider an Eulerian cycle which starts with $\perp_T$ and ends with the edge $(\mathsf{child}_1(\perp_T), \perp_T)$, and for any $1 \le i \le n$, let ℓ_i be the i-th leaf to pass in the cycle, where $n = |V_T^{leaf}|$ (note that the cycle contains each leaf just one time). Let w be a string of length n such that for any $1 \le i \le n$, $w[i] = g((\perp_T, v))$ with $(\perp_T, v) \in E_T$ and $\ell_i \in V_T^{leaf}(v)$. Then we prove that w realizes (T, f) and g.
Let $(v_p, v) \in E_T$ and $\ell_i, \ell_j \in V_T^{leaf}(v)$. It follows from the definition of E_G that both ℓ_{i+1} and ℓ_{j+1} are in $V_T^{leaf}(u)$, where $u \in \mathsf{children}(f(v))$ such that $g((f(v), u)) = g((v_p, v))$. Recursively applying this $h = sldepth(v_p)$ times, we get $w[i : i + h] = w[j : j + h]$ and $w[i + h] = w[j + h] = g((v_p, v))$. Here let us associate v with $w[i : i + h]$. Then for any $v \in V_T$, the substring associated with v occurs in w exactly $|V_T^{leaf}(v)|$ times, and w contains no other substring. Hence, w realizes (T, f) and g.

only if part: Let w be a string of length n which realizes (T, f) and g. Since STG_g is an Eulerian graph, it suffices to show that $\perp_T$ and all leaves of T are connected by some path on STG_g.
For any $1 \le i < n$, let u and v be the leaf node corresponding to $w[i : n]$ and $w[i + 1 : n]$, respectively. Let v_1 be the node such that $(u, v_1) \in E_G$. Evidently

$v \in V_T^{leaf}(v_1)$. Then we prove that $|V_T^{leaf}(v_j)| > D_g(v_j)$ for any $1 < j \leq k$, where $v_k = v$ and $\mathsf{par}(v_j) = v_{j-1}$ for any $1 < j \leq k$. Assume on the contrary that $|V_T^{leaf}(v_j)| = D_g(v_j)$ for some $1 < j \leq k$. It means that ax (a is any character in $\Sigma - \{w[i]\}$) occurs $|V_T^{leaf}(v_j)| = D_g(v)$ times in w, where x is a string corresponding to v_j. This contradicts that $w[i]x$ occurs in w. Consequently there is a path from u to v, and hence, all leaves are connected by some path. In addition, it is clear that $\perp_T$ and $\mathsf{child}_1(\perp_T)$ (the node related to $w[n:n]$) is connected by $(\mathsf{child}_1(\perp_T), \perp_T)$. Therefore STG_g has an Eulerian cycle which contains $\perp_T$ and all leaves of T. □

Lemma 13. *For any input (T, f) of Problem 1 and any labeling function g,*

1. *we can check whether or not there exists a string w which realizes (T, f) and g,*
2. *we can compute a string which realizes (T, f) and g, if such exists,*

in linear time in the size of T.

Proof. First of all, if $|V_T^{leaf}(v)| < D_g(v)$ for some $v \in V_T$, then g is invalid. Otherwise, we consider an Eulerian cycle on STG_g. If a cycle which contains $\perp_T$ and all leaves of T is found, we can construct a string which realizes (T, f) and g as discussed in Lemma 12. If no such cycles are found, then g is invalid. Since an Eulerian cycle in a given Eulerian graph can be computed in linear time in the size of the graph, it follows from Lemma 11 that these operations can be done in $O(|V_T|)$ time. □

3.2 Linear Time Algorithm for a Binary Alphabet

The following theorem is the main result of this paper.

Theorem 14. *On a binary alphabet, Problem 1 can be solved in linear time.*

Proof. Let (T, f) be an input of Problem 1. By Lemma 13, given a valid labeling, we can construct a string which realizes (T, f). Then the remaining task is to search for a valid labeling.

In the binary case, the following conditions are needed for (T, f) to be valid:

- For any $v \in V_T^{in}$, the number of children of v is 2 or 3.
- For any $v \in V_T^{in}$, if $|\mathsf{children}(v)| = 3$, the first child of v is a leaf node.
- For any $v \in (V_T^{in} - \{\perp_T\})$, if $|\mathsf{children}(v)| = 3$, then $|\mathsf{children}(f(v))| = 3$.

Any input that does not satisfy these conditions is filtered out in the process (discussed in Subsection 3.1) of checking whether or not there exists g which satisfies Preconditions 1, 2 and 3. Also, remark that the checking process uniquely determines the label $g((v_p, v))$ for any $(v_p, v) \in E_T$ with $v_p = \perp_T$ or $v \in V_T^{in}$.

Let $\Sigma = \{\mathsf{a}, \mathsf{b}\}$. For any $v \in V_T^{in}$ with $|\mathsf{children}(v)| = |\mathsf{children}(f(v))|$, the label $g((v, \mathsf{child}_i(v))$ is fixed to $g((f(v), \mathsf{child}_i(f(v)))$ for any $1 \leq i \leq |\mathsf{children}(v)|$. Hence, the degree of freedom to determine a labeling g lies only in nodes $v \in V_T^{in}$ such that $|\mathsf{children}(v)| = 2$ and $|\mathsf{children}(f(v))| = 3$. Moreover, by Lemma 8, we need to choose g with $|V_T^{leaf}(v)| \geq D_g(v)$ for any $v \in V_T$. In particular, for the binary case, the next arguments hold:

- If there exist $u, v \in V_T^{in}$ with $u \neq v$, $f(u) = f(v) = q$ and $|\mathsf{children}(u)| = |\mathsf{children}(v)| = |\mathsf{children}(q)| = 3$, then the input (T, f) is invalid. This is because, if such exists, then it leads to $|V_T^{leaf}(q')| = 1 < 2 \leq D_g(q')$, where $q' = \mathsf{child}_1(q)$.

- Let $q \in V_T^{in}$ such that $|\mathsf{children}(q)| = 3$ and there is not $v \in V_T^{in}$ with $|\mathsf{children}(v)| = 3$ and $f(v) = q$. Note that such a node is unique if the above condition holds. Then, it follows from Corollary 6 that $|\{v \in V_T^{in} \mid f(v) = q, |\mathsf{children}(v)| = 2\}| \leq 2$.
- For any $v \in V_T^{in}$ with $|\mathsf{children}(v)| = 2$ and $|\mathsf{children}(f(v))| = 3$, if there exists $u \in V_T^{in}$ with $f(u) = f(v)$, $|\mathsf{children}(u)| = 3$ and $|\mathsf{children}(f(u))| = 3$, then $g((v, \mathsf{child}_1(v))) = \mathtt{a}$ and $g((v, \mathsf{child}_2(v))) = \mathtt{b}$.

Putting these together, we see that the number of possible labelings is at most 5, namely, in the maximum case, there are the following five possible allocations

$$\begin{aligned}&\langle g(u, \mathsf{child}_1(u)), g(u, \mathsf{child}_2(u)), g(v, \mathsf{child}_1(v)), g(v, \mathsf{child}_2(v))\rangle \\ &\in \{\langle \mathtt{a}, \mathtt{b}, \mathtt{a}, \mathtt{b}\rangle, \langle \$, \mathtt{a}, \mathtt{a}, \mathtt{b}\rangle, \langle \$, \mathtt{b}, \mathtt{a}, \mathtt{b}\rangle, \langle \mathtt{a}, \mathtt{b}, \$, \mathtt{a}\rangle, \langle \mathtt{a}, \mathtt{b}, \$, \mathtt{b}\rangle\},\end{aligned}$$

where $u, v \in V_T^{in}$ such that $u \neq v$, $f(u) = f(v) = q$, $|\mathsf{children}(u)| = |\mathsf{children}(v)| = 2$ and $|\mathsf{children}(q)| = 3$.

Figure 8 illustrates all the possible labelings to the input shown in Figure 2.

Thus, we only have to check at most five labeling functions. It follows from Lemma 13 that each labeling can be checked in $O(|V_T|)$ time, and hence, Problem 1 can be solved in linear time in the input size. □

4 Conclusions and Future Work

We presented a linear time algorithm to solve the reverse engineering problem on suffix trees for a binary alphabet, where inner nodes of input suffix trees are augmented with suffix links. The algorithm is designed based on combinatorial properties of suffix trees. There remain a lot to do on the reverse engineering problem of suffix trees. For instance: Can we enumerate all strings that realize a given unlabeled tree? Can we solve the problem for larger alphabets? Can we solve the problem without suffix links?

References

1. A. Apostolico: *The myriad virtues of subword trees.* Combinatorial Algorithms on Words, F12 1985, pp. 85–96.
2. H. Bannai, S. Inenaga, A. Shinohara, and M. Takeda: *Inferring strings from graphs and arrays*, in Proc. MFCS 2003, vol. 2747 of LNCS, 2003, pp. 208–217.
3. J. Clément, M. Crochemore, and G. Rindone: *Reverse engineering prefix tables*, in Proc. STACS 2009, 2009, pp. 289–300.
4. M. Crochemore, C. Iliopoulos, S. Pissis, and G. Tischler: *Cover array string reconstruction*, in Proc. CPM 2010, vol. 6129 of LNCS, 2010, pp. 251–259.
5. J.-P. Duval, T. Lecroq, and A. Lefebvre: *Border array on bounded alphabet.* Journal of Automata, Languages and Combinatorics, 10(1) 2005, pp. 51–60.
6. J.-P. Duval, T. Lecroq, and A. Lefebvre: *Efficient validation and construction of border arrays and validation of string matching automata.* RAIRO - Theoretical Informatics and Applications, 43(2) 2009, pp. 281–297.
7. J.-P. Duval and A. Lefebvre: *Words over an ordered alphabet and suffix permutations.* Theoretical Informatics and Applications, 36 2002, pp. 249–259.
8. F. Franek, S. Gao, W. Lu, P. J. Ryan, W. F. Smyth, Y. Sun, and L. Yang: *Verifying a border array in linear time.* J. Comb. Math. and Comb. Comp., 42 2002, pp. 223–236.
9. P. Gawrychowski, A. Jez, and L. Jez: *Validating the Knuth-Morris-Pratt failure function, fast and online*, in Proc. CSR 2010, vol. 6072 of LNCS, 2010, pp. 132–143.
10. D. Gusfield: *Algorithms on Strings, Trees, and Sequences*, Cambridge University Press, New York, 1997.

11. T. I, S. Inenaga, H. Bannai, and M. Takeda: *Counting parameterized border arrays for a binary alphabet*, in Proc. LATA 2009, vol. 5457 of LNCS, 2009, pp. 422–433.
12. T. I, S. Inenaga, H. Bannai, and M. Takeda: *Counting and verifying maximal palindromes*, in Proc. SPIRE 2010, vol. 6393 of LNCS, 2010, pp. 135–146.
13. T. I, S. Inenaga, H. Bannai, and M. Takeda: *Verifying a parameterized border array in $O(n^{1.5})$ time*, in Proc. CPM 2010, vol. 6129 of LNCS, 2010, pp. 238–250.
14. W. Matsubara, A. Ishino, and A. Shinohara: *Inferring strings from runs*, in Proc. PSC 2010, 2010, pp. 150–160.
15. E. M. McCreight: *A space-economical suffix tree construction algorithm.* Journal of the ACM, 23(2) 1976, pp. 262–272.
16. D. Moore, W. F. Smyth, and D. Miller: *Counting distinct strings.* Algorithmica, 23(1) 1999, pp. 1–13.
17. K.-B. Schürmann and J. Stoye: *Counting suffix arrays and strings.* Theoretical Computer Science, 395(2-3) 2008, pp. 220–234.
18. E. Ukkonen: *On-line construction of suffix trees.* Algorithmica, 14(3) 1995, pp. 249–260.
19. P. Weiner: *Linear pattern-matching algorithms*, in Proc. of 14th IEEE Ann. Symp. on Switching and Automata Theory, 1973, pp. 1–11.

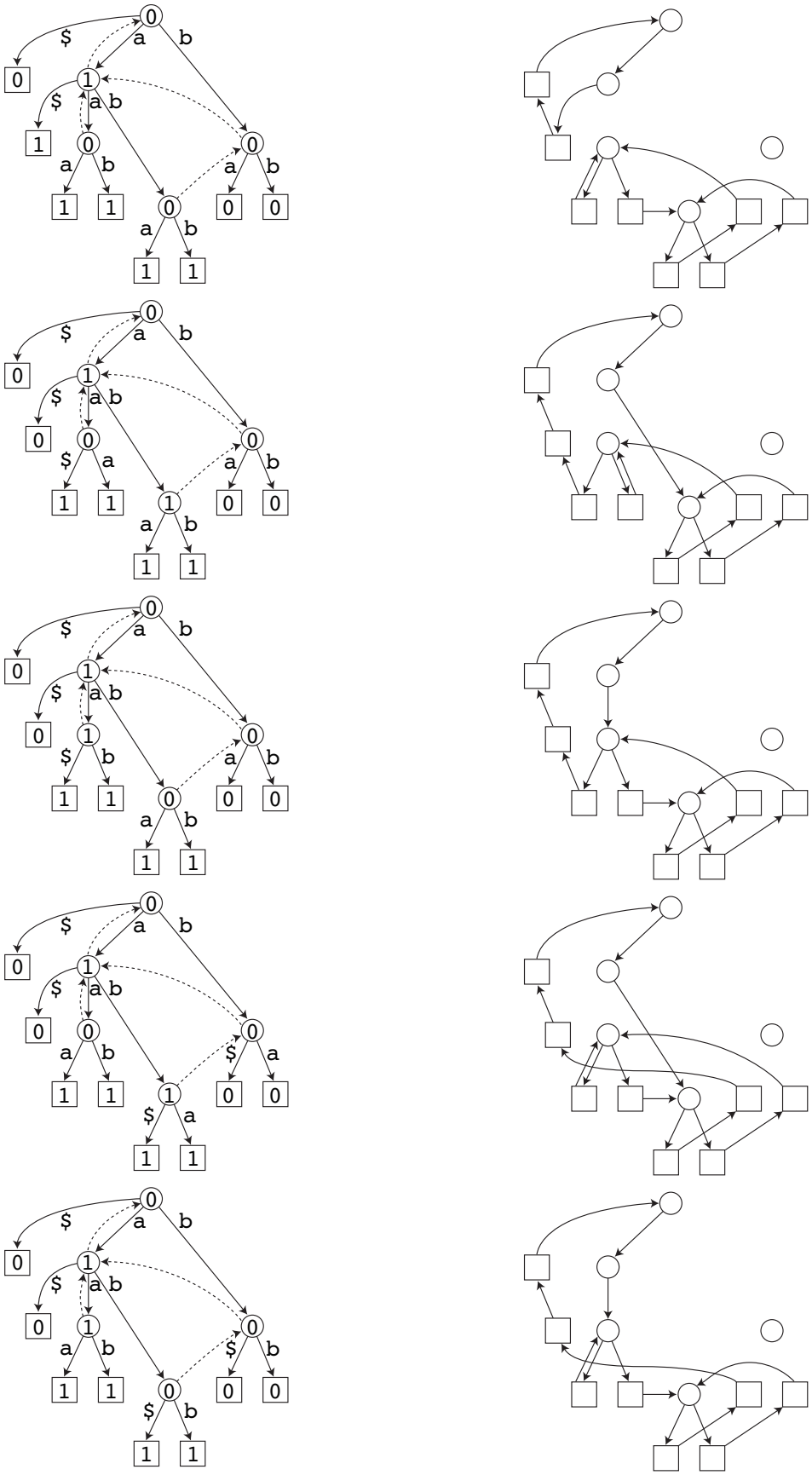

Figure 8. All the possible labelings. The top one is invalid and the others are valid.