Bidirectional Construction of Suffix Trees

Shunsuke Inenaga

Department of Informatics, Kyushu University 33, Fukuoka 812-8581, Japan

e-mail: `s-ine@i.kyushu-u.ac.jp`

Abstract. String matching is critical in information retrieval since in many cases information is stored and manipulated as strings. Constructing and utilizing suitable data structures for text strings, we can solve the string matching problem efficiently. Such structures are called *index structures*. The *suffix tree* is certainly the most widely-known and extensively-studied structure of this kind. In this paper, we present a linear-time algorithm for bidirectional construction of suffix trees.

1 Introduction

Pattern matching on strings is of central importance to Theoretical Computer Science. The pattern matching problem is to examine whether a given pattern string p matches a text string w. This problem can be solved in $O(|p|)$ time, by using a suitable *index structure*.

The most basic index structure seems to be the suffix trie, by whose nodes all substrings of a given string w are recognized. Probably the structure is the easiest to understand, but its only, however biggest drawback is that its space requirement is $O(|w|^2)$.

This fact led the introduction of more space-economical ($O(|w|)$-spaced) structures such as the suffix tree [23, 19, 22, 12], the directed acyclic word graph (DAWG) [3, 7, 2], the compact directed acyclic word graph (CDAWG) [4, 9, 15, 13, 16], the suffix array [18], and some other variants. Among those, suffix trees are possibly most widely-known and extensively-studied [8, 12], perhaps because there are a 'myriad' [1] of applications for them.

Construction of suffix trees has been considered in various contexts: Weiner [23] invented the first algorithm that constructs suffix trees in linear time; McCreight [19] proposed a more space-economical algorithm than Weiner's; Chen and Seiferas [6] showed an efficient modification of Weiner's algorithm; Ukkonen [22] introduced an on-line algorithm to construct suffix trees, which Giegerich and Kurtz [11] regarded as "the most elegant"; Farach [10] considered optimal construction of suffix trees with large alphabets; Breslauer [5] gave a linear-time algorithm for building the suffix tree of a given trie that stores a set of strings; Inenaga et al. [14] presented an on-line algorithm that simultaneously constructs both the suffix tree of a string and the DAWG of the reversed string.

In this paper we explore *bidirectional construction* of suffix trees. Namely, the algorithm we propose allows us to update the suffix tree of a string w to the suffix tree of a string xwy, where x, y are any strings. We also show that our algorithm runs in linear time and space with respect to the length of a given string.

Some related work can be seen in literature: Stoye [20, 21] invented variant of suffix trees, called *affix trees*. He proposed an algorithm for bidirectional construction of affix trees, and Maaß [17] improved the time complexity of the algorithm to $O(|w|)$.

2 Suffix Trees

Let Σ be a finite alphabet. An element of Σ^* is called a *string*. Strings x, y, and z are said to be a *prefix*, *factor*, and *suffix* of string $w = xyz$, respectively. The sets of prefixes, factors, and suffixes of a string w are denoted by $\mathit{Prefix}(w)$, $\mathit{Factor}(w)$, and $\mathit{Suffix}(w)$, respectively. The length of a string w is denoted by $|w|$. The empty string is denoted by ε, that is, $|\varepsilon| = 0$. Let $\Sigma^+ = \Sigma^* - \{\varepsilon\}$. The i-th character of a string w is denoted by $w[i]$ for $1 \leq i \leq |w|$. Let $S \subseteq \Sigma^*$. The cardinality of S is denoted by $|S|$. For any string $u \in \Sigma^*$, $Su^{-1} = \{x \mid xu \in S\}$.

Let $w \in \Sigma^*$. We define an equivalence relation $\equiv^L_w$ on Σ^* by

$$x \equiv^L_w y \Leftrightarrow \mathit{Prefix}(w)x^{-1} = \mathit{Prefix}(w)y^{-1}.$$

The equivalence class of a string $x \in \Sigma^*$ with respect to $\equiv^L_w$ is denoted by $[x]^L_w$. Note that all strings not belonging to $\mathit{Factor}(w)$ form one equivalence class under $\equiv^L_w$. This equivalence class is called the *degenerate* class. All other classes are said to be *non-degenerate*.

Proposition 1 ([14]) *Let $w \in \Sigma^*$ and $x, y \in \mathit{Factor}(w)$. If $x \equiv^L_w y$, then either x is a prefix of y, or vice versa.*

Proof. By the definition of $\equiv^L_w$, we have $\mathit{Prefix}(w)x^{-1} = \mathit{Prefix}(w)y^{-1}$. There are three cases to consider:

(1) When $|x| = |y|$. Obviously, $x = y$ in this case. Thus $x \in \mathit{Prefix}(y)$ and $y \in \mathit{Prefix}(x)$.

(2) When $|x| > |y|$. Let u be an arbitrary string in $\mathit{Prefix}(w)$. Assume $u = sx$ with $s \in \Sigma^*$. Then $s \in \mathit{Prefix}(w)x^{-1}$, which results in $s \in \mathit{Prefix}(w)y^{-1}$. Hence, there must exist a string $v \in \mathit{Prefix}(w)$ such that $v = sy$. By the assumption that $|x| > |y|$, we have $|u| > |v|$. From the fact that both u and v are in $\mathit{Prefix}(w)$, it is derived that $v \in \mathit{Prefix}(u)$. Consequently, $y \in \mathit{Prefix}(x)$.

(3) When $|x| < |y|$. By a similar argument to the one in Case (2), we have $x \in \mathit{Prefix}(y)$.

□

For any string $x \in \mathit{Factor}(w)$, the longest member in $[x]^L_w$ is denoted by $\overset{w}{\overrightarrow{x}}$.

Proposition 2 ([14]) *Let $w \in \Sigma^*$. For any $x \in \mathit{Factor}(w)$, there uniquely exists a string $\alpha \in \Sigma^*$ such that $\overset{w}{\overrightarrow{x}} = x\alpha$.*

Proof. Let $\overset{w}{\overrightarrow{x}} = x\alpha$ with $\alpha \in \Sigma^*$. For the contrary, assume there exists a string $\beta \in \Sigma^*$ such that $\overset{w}{\overrightarrow{x}} = x\beta$ and $\beta \neq \alpha$. By Proposition 1, either $x\alpha \in \mathit{Prefix}(x\beta)$ or $x\beta \in \mathit{Prefix}(x\alpha)$ must stand, since $x\alpha \equiv^L_w x\beta$. However, neither of them actually holds since $|\alpha| = |\beta|$ and $\alpha \neq \beta$, which yields a contradiction. Hence, α is the only string satisfying $\overset{w}{\overrightarrow{x}} = x\alpha$. □

Proposition 3 *Let $w \in \Sigma^*$ and $x \in \mathit{Factor}(w)$. Assume $\overset{w}{\overrightarrow{x}} = x$. Then, for any $y \in \mathit{Suffix}(x)$, $\overset{w}{\overrightarrow{y}} = y$.*

Proof. Assume contrarily that there uniquely exists a string $\alpha \in \Sigma^+$ such that $\overset{w}{\overrightarrow{y}} = y\alpha$. Since $y \in \mathit{Suffix}(x)$, x is always followed by α in w. It implies that $\mathit{Prefix}(w)x^{-1} = \mathit{Prefix}(w)(x\alpha)^{-1}$, and therefore we have $x \equiv^L_w x\alpha$. That $|\alpha| > 0$ means that $\overset{w}{\overrightarrow{x}}$ is not the longest in $[x]^L_w$; a contradiction. Hence, $\overset{w}{\overrightarrow{y}} = y$. □

Proposition 4 *Let $w \in \Sigma^*$. For any string $x \in \mathit{Suffix}(w)$, $\overset{w}{\overrightarrow{x}} = x$.*

Proof. Let $y \in \Sigma^*$ be an arbitrary string such that $x \equiv^L_w y$ and $x \neq y$. Then, we have $\mathit{Prefix}(w)x^{-1} = \mathit{Prefix}(w)y^{-1}$. Because $x \in \mathit{Suffix}(w)$, $y \in \mathit{Prefix}(x) - \{x\}$ and thus $|x| > |y|$. Hence, $\overset{w}{\overrightarrow{x}} = x$. □

The number of strings in $\mathit{Factor}(w)$ is $O(|w|^2)$. For example, consider string $a^n b^n$. However, for any string $w \in \Sigma^*$, the number of strings x such that $x = \overset{w}{\overrightarrow{x}}$ is $O(|w|)$. The following lemma gives a tighter upperbound.

Lemma 1 ([3, 4]) *Assume that $|w| > 1$. The number of the non-degenerate equivalence classes in $\equiv^L_w$ is at most $2|w| - 1$.*

In the following, we define the suffix tree of a string $w \in \Sigma^*$, denoted by $\mathit{STree}(w)$, on the basis of the above-mentioned equivalence classes. We define it as an edge-labeled tree (V, E) with $E \subseteq V \times \Sigma^+ \times V$ where the second component of each edge represents its label. We also give a definition of the *suffix links*, kinds of failure functions, frequently utilized for time-efficient construction of suffix trees [23, 19, 22].

Definition 1 *$\mathit{STree}(w)$ is the tree (V, E) such that*

$$\begin{aligned} V &= \{\overset{w}{\overrightarrow{x}} \mid x \in \mathit{Factor}(w)\}, \\ E &= \{(\overset{w}{\overrightarrow{x}}, a\beta, \overset{w}{\overrightarrow{xa}}) \mid x, xa \in \mathit{Factor}(w),\ a \in \Sigma,\ \beta \in \Sigma^*,\ \overset{w}{\overrightarrow{xa}} = xa\beta, \text{ and } \overset{w}{\overrightarrow{x}} \neq \overset{w}{\overrightarrow{xa}}\}, \end{aligned}$$

and its suffix links are the set

$$F = \{(\overset{w}{\overrightarrow{ax}}, \overset{w}{\overrightarrow{x}}) \mid x, xa \in \mathit{Factor}(w),\ a \in \Sigma, \text{ and } \overset{w}{\overrightarrow{ax}} = a \cdot \overset{w}{\overrightarrow{x}}\}.$$

The node $\overset{w}{\overrightarrow{\varepsilon}} = \varepsilon$ is called the *root* node of $\mathit{STree}(w)$. When a node $\overset{w}{\overrightarrow{x}}$ is of out-degree zero, it is said to be a *leaf* node. Each leaf node corresponds to a string in $\mathit{Suffix}(w)$. If $x \in \mathit{Factor}(w)$ satisfies $x = \overset{w}{\overrightarrow{x}}$, x is said to be represented on *explicit* node $\overset{w}{\overrightarrow{x}}$. If $x \neq \overset{w}{\overrightarrow{x}}$, x is said to be on an *implicit* node. STree(coco) and STree(cocoa) are displayed in Figure 1.

It derives from Lemma 1 that:

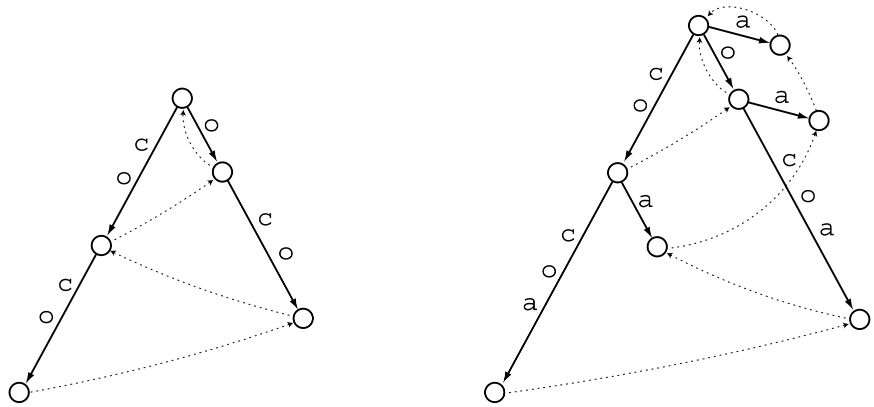

Figure 1: $STree$(`coco`) on the left, and $STree$(`cocoa`) on the right. Solid arrows represent edges, while dotted arrows denote suffix links.

Theorem 1 ([19]) *Let $w \in \Sigma^*$. Let $STree(w) = (V, E)$. Assume $|w| > 1$. Then $|V| \leq 2|w| - 1$ and $|E| \leq 2|w| - 2$.*

Weiner's algorithm [23] and McCreight's algorithm [19] construct the suffix tree defined above, $STree(w)$. On the other hand, Ukkonen's algorithm constructs a slightly different version, which is suitable for his algorithm.

As a preliminary to define the modified suffix tree, we firstly introduce a relation X_w over Σ^* such that

$$X_w = \{(x, xa) \mid x \in Factor(w) \text{ and } a \in \Sigma \text{ is unique such that } xa \in Factor(w)\}.$$

Let $\equiv'^L_w$ be the equivalence closure of X_w, i.e., the smallest superset of X_w that is symmetric, reflexive, and transitive.

Proposition 5 ([14]) *For any string $w \in \Sigma^*$, $\equiv^L_w$ is a refinement of $\equiv'^L_w$.*

Proof. Let x, y be any strings in $Factor(w)$ and assume $x \equiv^L_w y$. According to Proposition 1, we firstly assume that $x \in Prefix(y)$. It follows from Proposition 2 that there uniquely exist strings $\alpha, \beta \in \Sigma^*$ such that $\overset{w}{\overrightarrow{x}} = x\alpha$ and $\overset{w}{\overrightarrow{y}} = y\beta$. Note that $\beta \in Suffix(\alpha)$. Let $\gamma \in \Sigma^*$ be the string satisfying $\alpha = \gamma\beta$. Then γ is the sole string such that $x\gamma = y$. By the definition of $\equiv'^L_w$, we have $x \equiv'^L_w y$. A similar argument holds in case that $y \in Prefix(x)$. □

Corollary 1 ([14]) *For any string $w \in \Sigma^*$, every equivalence class under $\equiv'^L_w$ is a union of one or more equivalence classes under $\equiv^L_w$.*

For a string $x \in Factor(w)$, the longest string in the equivalence class with respect to x under $\equiv'^L_w$ is denoted by $\overset{\overset{w}{\Longrightarrow}}{x}$.

The next proposition corresponds to Proposition 3

Proposition 6 *Let $w \in \Sigma^*$ and $x \in Factor(w) - Suffix(w)$. Assume $\overset{\overset{w}{\Longrightarrow}}{x} = x$. Then, for any $y \in Suffix(x)$, $\overset{\overset{w}{\Longrightarrow}}{y} = y$.*

Proof. Since $\overset{w}{\overset{\Longrightarrow}{x}} = x$ and $x \notin \mathit{Suffix}(w)$, there are at least two characters $a, b \in \Sigma$ such that $xa, xb \in \mathit{Factor}(w)$ and $a \neq b$. Since $y \in \mathit{Suffix}(x)$, y is also followed by both a and b in the string w. Thus $\overset{w}{\overset{\Longrightarrow}{y}} = y$. $\square$

Remark that the precondition of the above proposition slightly differs from that of Proposition 3. Namely, when x is a suffix of w, this proposition does not always hold.

From here on, we explore some relationship between $\overset{w}{\overrightarrow{(\cdot)}}$ and $\overset{w}{\overset{\Longrightarrow}{(\cdot)}}$.

Lemma 2 ([14]) *Let $w \in \Sigma^*$. For any string $x \in \mathit{Factor}(w)$, $\overset{w}{\overrightarrow{x}}$ is a prefix of $\overset{w}{\overset{\Longrightarrow}{x}}$. If $\overset{w}{\overrightarrow{x}} \neq \overset{w}{\overset{\Longrightarrow}{x}}$, then $\overset{w}{\overrightarrow{x}} \in \mathit{Suffix}(w)$.*

Proof. We can prove that $\overset{w}{\overrightarrow{x}} \in \mathit{Prefix}(\overset{w}{\overset{\Longrightarrow}{x}})$ by Proposition 1 and Corollary 1. Now suppose $\overset{w}{\overrightarrow{x}} \neq \overset{w}{\overset{\Longrightarrow}{x}}$. Let $\overset{w}{\overrightarrow{x}} = x\beta$ with $\beta \in \Sigma^+$. Supposing $\overset{w}{\overset{\Longrightarrow}{x}} = x\alpha$ with $\alpha \in \Sigma^+$, we have $\beta \in \mathit{Prefix}(\alpha)$. Let $\beta\gamma = \alpha$ with $\gamma \in \Sigma^*$. By the assumption $\overset{w}{\overrightarrow{x}} \neq \overset{w}{\overset{\Longrightarrow}{x}}$, we have $x\beta \not\equiv^L_w x\alpha$, although γ is the sole string that follows $x\beta$ in w since $\overset{w}{\overset{\Longrightarrow}{x}} = x\alpha$. Therefore, x must be a suffix of w, which is followed by *no* character. $\square$

For example, consider string $w = \texttt{coco}$. Then, $\overset{w}{\overrightarrow{\texttt{co}}} = \texttt{co}$ but $\overset{w}{\overset{\Longrightarrow}{\texttt{co}}} = \texttt{coco}$, where $\texttt{co}$ is a suffix of $\texttt{coco}$.

Lemma 3 *Let $w \in \Sigma^*$ and $x \in \mathit{Suffix}(w)$. If $x \notin \mathit{Prefix}(y)$ for any string $y \in \mathit{Factor}(w) - \{x\}$, then $\overset{w}{\overrightarrow{x}} = \overset{w}{\overset{\Longrightarrow}{x}}$.*

Proof. The precondition implies that there is no character $a \in \Sigma$ satisfying $xa \in \mathit{Factor}(w)$. Thus we have $\overset{w}{\overset{\Longrightarrow}{x}} = x$. On the other hand, we obtain $\overset{w}{\overrightarrow{x}} = x$ by Proposition 4, because $x \in \mathit{Suffix}(w)$. Hence $\overset{w}{\overrightarrow{x}} = \overset{w}{\overset{\Longrightarrow}{x}}$. $\square$

Lemma 4 *Let $w \in \Sigma^*$ with $|w| = n$. Assume that the last character $w[n]$ is unique in w, that is, $w[n] \neq w[i]$ for any $1 \leq i \leq n-1$. Then, for any string $x \in \mathit{Factor}(w)$, $\overset{w}{\overrightarrow{x}} = \overset{w}{\overset{\Longrightarrow}{x}}$.*

Proof. By the contraposition of the second statement of Lemma 2, if $x \notin \mathit{Suffix}(w)$, then $\overset{w}{\overrightarrow{x}} = \overset{w}{\overset{\Longrightarrow}{x}}$. Because of the unique character $w[n]$, any suffix z of w satisfies the precondition of Lemma 3, and thus $\overset{w}{\overrightarrow{z}} = \overset{w}{\overset{\Longrightarrow}{z}}$. $\square$

We are now ready to define $\mathit{STree}'(w)$, which is a modified version of $\mathit{STree}(w)$.

Definition 2 *$\mathit{STree}'(w)$ is the tree (V, E) such that*

$$\begin{aligned} V &= \{\overset{w}{\overset{\Longrightarrow}{x}} \mid x \in \mathit{Factor}(w)\}, \\ E &= \{(\overset{w}{\overset{\Longrightarrow}{x}}, a\beta, \overset{w}{\overset{\Longrightarrow}{xa}}) \mid x, xa \in \mathit{Factor}(w),\ a \in \Sigma,\ \beta \in \Sigma^*,\ \overset{w}{\overset{\Longrightarrow}{xa}} = xa\beta, \text{ and } \overset{w}{\overset{\Longrightarrow}{x}} \neq \overset{w}{\overset{\Longrightarrow}{xa}}\}, \end{aligned}$$

and its suffix links are the set

$$F = \{(\overset{w}{\overset{\Longrightarrow}{ax}}, \overset{w}{\overset{\Longrightarrow}{x}}) \mid x, xa \in \mathit{Factor}(w),\ a \in \Sigma,\ \text{and } \overset{w}{\overset{\Longrightarrow}{ax}} = a \cdot \overset{w}{\overset{\Longrightarrow}{x}}\}.$$

Remark that $STree'(w)$ can be obtained by replacing $\overset{\overset{w}{\longrightarrow}}{(\cdot)}$ in $STree(w)$ with $\overset{\overset{w}{\Longrightarrow}}{(\cdot)}$.

We have the next lemma deriving from Lemma 4.

Lemma 5 *Let $w \in \Sigma^*$ with $|w| = n$. Assume that the last character $w[n]$ is unique in w, that is, $w[n] \neq w[i]$ for any $1 \leq i \leq n-1$. Then, $STree(w) = STree'(w)$.*

For comparing $STree(w)$ and $STree'(w)$, see Figure 1 and Figure 2. As shown in Proposition 3, any suffixes of a string represented by an explicit node are also explicit.

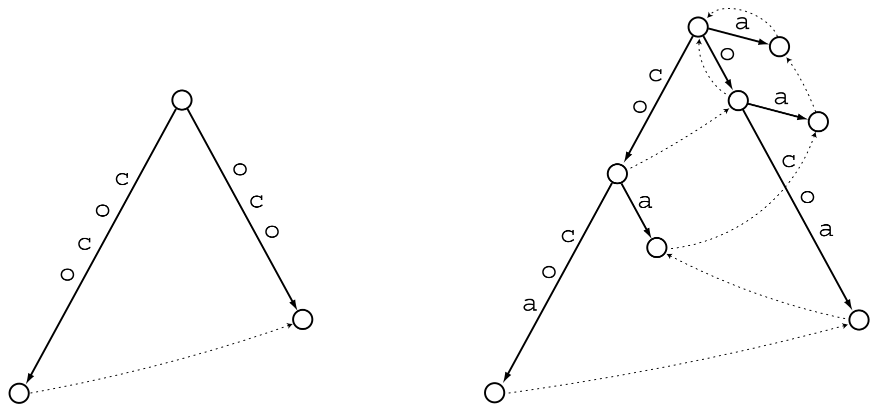

Figure 2: $STree'$(coco) on the left, and $STree'$(cocoa) on the right. Solid arrows represent the edges, while dotted arrows denote suffix links.

According to Lemma 5, using a delimiter \$ that occurs nowhere in w, we have $STree(w\$) = STree'(w\$)$ for any $w \in \Sigma^*$.

3 Bidirectional Construction of Suffix Trees

3.1 Right Extension

Assume that we have $STree'(w)$ with some $w \in \Sigma^*$. Now we consider updating it into $STree'(wa)$ with $a \in \Sigma$, by inserting the suffixes of wa into $STree'(w)$. Ukkonen [22] achieved the following result.

Theorem 2 ([22]) *For any $a \in \Sigma$ and $w \in \Sigma^*$, $STree'(w)$ can be updated to $STree'(wa)$ in amortized constant time.*

Here we only recall essence of Ukkonen's algorithm together with some supporting lemmas and propositions.

Let y be the longest string in $Factor(w) \cap Suffix(wa)$. Then y is called the *longest repeated suffix* of wa and denoted by $LRS(wa)$. Since every string $x \in Suffix(y)$ belongs to $Factor(w)$, we do not need to newly insert any x into $STree'(w)$.

Lemma 6 *Let $a \in \Sigma$ and $w \in \Sigma^*$. Let $y = LRS(w)$. For any string $x \in Suffix(w) - Suffix(y)$, $\overset{\overset{wa}{\Longrightarrow}}{x} = \overset{\overset{w}{\Longrightarrow}}{x} \cdot a$.*

Proof. Since $y = LRS(w)$, any string $x \in \mathit{Suffix}(w) - \mathit{Suffix}(y)$ appears only once in w as a suffix of w, and is therefore $\overset{w}{\overset{\Rightarrow}{x}} = x$. Also, x is followed only by a in wa, and thus $\overset{wa}{\overset{\Rightarrow}{x}} = xa$. $\square$

The above lemma implies that a leaf node of $STree'(w)$ is also a leaf node in $STree'(wa)$. Thus we need no explicit maintenance for leaf nodes. Namely, we can insert all strings of $\mathit{Suffix}(w) - \mathit{Suffix}(y)$ into $STree'(w)$ *automatically* (for more detail, see [22]).

Proposition 7 *Let $a \in \Sigma$ and $w \in \Sigma^*$. Let $y = LRS(w)$ and $z = LRS(wa)$. For any string $x \in \mathit{Suffix}(y) - \mathit{Suffix}(z)a^{-1}$, $\overset{wa}{\overset{\Rightarrow}{x}} = x$.*

Proof. Firstly, we consider the empty string ε. It always belongs to $\mathit{Suffix}(y) - \mathit{Suffix}(z)a^{-1}$, since $\varepsilon \in \mathit{Suffix}(y)$ and $\varepsilon \notin \mathit{Suffix}(z)a^{-1}$. It is now obvious that $\overset{wa}{\overset{\Rightarrow}{\varepsilon}} = \varepsilon$. Now we consider other strings. That $xa \notin \mathit{Suffix}(z)$ implies the existence of $b \in \Sigma$ such that $xb \in \mathit{Factor}(w)$ and $b \neq a$. Therefore, we have $\overset{wa}{\overset{\Rightarrow}{x}} = x$. $\square$

We start from the location corresponding to $LRS(w)$ and convert $STree'(w)$ to $STree'(wa)$, while creating new explicit nodes if necessary to insert new suffixes into $STree'(w)$, according to the above proposition. Now the next question is how to detect the locations where new explicit nodes should be created.

We here define the *eliminator* ξ for any character $a \in \Sigma$ by

$$a\xi = \xi a = \varepsilon$$

and $|\xi| = -1$. Moreover, we define that $\xi \in \mathit{Prefix}(\varepsilon)$ and $\xi \in \mathit{Suffix}(\varepsilon)$, but $\xi \notin \mathit{Prefix}(x)$ and $\xi \notin \mathit{Suffix}(x)$ for any $x \in \Sigma^+$. The symbol ξ corresponds to the auxiliary node $\perp$ introduced by Ukkonen [22]. Owing to the introduction of ξ, we can establish the following lemma.

Lemma 7 *Let $a \in \Sigma$ and $w \in \Sigma^*$. Let $y = LRS(w)$ and $z = LRS(wa)$. Let $x \in \mathit{Suffix}(y) - \mathit{Suffix}(z)a^{-1}$. Suppose t is the longest string in $\mathit{Prefix}(x)$ such that $\overset{w}{\overset{\Rightarrow}{t}} = t$. Let $x' = \mathit{Suffix}(x)$ with $|x'| + 1 = |x|$ and $t' = \mathit{Suffix}(t)$ with $|t'| + 1 = |t|$. For string $\alpha \in \Sigma^*$ such that $t\alpha = x$, $t'\alpha = x'$.*

Notice that we can reach string x' via the suffix link of the node for t in $STree'(w)$ and along the path spelling out α from the node for t' (recall Definition 2). Moreover, Proposition 6 guarantees that t' is an explicit node in $STree'(w)$. Ukkonen proved that x' can be found in amortized constant time by using the suffix link of node $\overset{w}{\overset{\Rightarrow}{t}}$.

3.2 Left Extension

Weiner [23] proposed an algorithm to construct $STree(aw)$ by updating $STree(w)$ with $a \in \Sigma$ in amortized constant time. On the other hand, this section is devoted to the exposition of the conversion from $STree'(w)$ to $STree'(aw)$. In so doing, we insert prefixes of aw into $STree'(w)$.

Lemma 8 *Let $a \in \Sigma$ and $w \in \Sigma^*$. For any string $x \in Factor(w) - Prefix(aw)$, $\overset{w}{\overrightarrow{x}} = \overset{aw}{\overrightarrow{x}}$.*

Proof. Let b be the unique character that follows x in w. (When $\overset{w}{\overrightarrow{x}} = x$, then $b = \varepsilon$.) Since $x \notin Prefix(aw)$, there is no new occurrence of x in aw. Therefore, b is also the only character following x in aw. Hence $\overset{w}{\overrightarrow{x}} = \overset{aw}{\overrightarrow{x}}$. □

The above lemma ensures that any implicit node of $STree'(w)$ does not become explicit in $STree'(aw)$ if it is not associated with any prefix of aw.

Now we turn our attention to the strings in $Prefix(aw)$. Let x be the longest string in set $Factor(w) \cap Prefix(aw)$. Then x is called the *longest repeated prefix* of aw and denoted by $LRP(aw)$. Since all prefixes of x belong to $Factor(w)$, we need not newly insert any of them into $STree'(w)$.

Proposition 8 *Let $a \in \Sigma$ and $w \in \Sigma^*$. Let $x = LRP(aw)$ and $y = LRS(w)$. If $x \notin Suffix(w) - Suffix(y)$, then $\overset{aw}{\overrightarrow{x}} = x$. Otherwise, $\overset{aw}{\overrightarrow{x}} = aw$.*

Proof. We first consider the case that $x \notin Suffix(w) - Suffix(y)$. Recall that x is the *longest* string in $Factor(w) \cap Prefix(aw)$. Moreover, $x \notin Suffix(w) - Suffix(y)$. Hence, there exist two characters $b, c \in \Sigma$ such that $xb, xc \in Factor(aw)$ and $b \neq c$. Thus we have $\overset{aw}{\overrightarrow{x}} = x$.

Now we consider the second case, $x \in Suffix(w) - Suffix(y)$. Here, x occurs only once in w as its suffix. Thus $\overset{w}{\overrightarrow{x}} = x$. On the other hand, by the definition of $LRP(aw)$, we obtain $x \in Prefix(aw) - \{aw\}$. Therefore, there uniquely exists a character $d \in \Sigma$ which follows x in aw. Hence we have $\overset{aw}{\overrightarrow{x}} = aw$. □

The above proposition implies that if $LRP(aw)$ is not on a leaf node in $STree'(w)$, it is represented by an explicit node in $STree'(aw)$, and otherwise it becomes implicit in $STree'(aw)$. We stress that this characterizes a difference between $STree'(w)$ and $STree(w)$. More concretely, Weiner's original algorithm constructs $STree(aw)$ on the basis of the next proposition.

Proposition 9 *For any $a \in \Sigma$ and $w \in \Sigma^*$, if $x = LRP(aw)$, then $\overset{aw}{\overrightarrow{x}} = x$.*

Now the next question is how to locate $LRP(aw)$ in $STree'(w)$. Our idea is similar to Weiner's strategy for constructing $STree(w)$ [23]. Let y be the longest element in set $Prefix(w) \cup \{\xi\}$ such that $ay \in Factor(w)$. Then y is called the *base* of aw and denoted by $Base(aw)$. On the other hand, let z be the longest element in set $Prefix(w) \cup \{\xi\}$ such that $\overset{w}{\overrightarrow{az}} = az$. Then z is called the *bridge* of aw and denoted by $Bridge(aw)$.

Lemma 9 ([23]) *Let $a \in \Sigma$ and $w \in \Sigma^*$. If $y = Base(aw)$, then $ay = LRP(aw)$.*

Proof. Assume contrarily that y' is the string such that $ay' = LRP(aw)$ and $|y'| > |y|$. By the definition of $LRP(aw)$, we have $ay' \in Prefix(aw)$, which yields $y' \in Prefix(w)$. It, however, contradicts the precondition that $y = Base(aw)$ since $|y'| > |y|$. □

According to the above lemma, we can utilize $Base(aw)$ for finding $LRP(aw)$ in $STree'(w)$.

Lemma 10 *Let $a \in \Sigma$ and $w \in \Sigma^*$. If $x = LRP(w)$, $y = Base(aw)$ and $z = Bridge(aw)$, then $y \in Prefix(x)$ and $z \in Prefix(y)$.*

Proof. By Lemma 9 we have $ay = LRP(aw)$. It is easy to see that $|LRP(w)| + 1 \geq |LRP(aw)|$, which implies $|x| \geq |y|$. Since $x, y \in Prefix(w)$, we obtain $y \in Prefix(x)$. It can be readily shown that $az \in Prefix(ay)$, since $ay = LRP(aw)$. Thus we have $z \in Prefix(y)$. □

The above lemma ensures that we can find both $Base(aw)$ and $Bridge(aw)$ by going up along the path from the node of $LRP(w)$ in $STree'(w)$.

Lemma 11 *Let $a \in \Sigma$ and $w \in \Sigma^*$. Let $y = Base(aw)$ and $z = Bridge(aw)$. Assume $\gamma \in \Sigma^*$ is the string satisfying $z\gamma = y$. Then, $az\gamma = LRP(aw)$.*

Proof. By Lemma 9 and Lemma 10. □

According to the above lemma, we can locate $LRP(aw)$ in $STree'(w)$ by going down from the node $\overset{w}{\overrightarrow{az}}$. The only thing not clarified yet is how to move from node $\overset{w}{\overrightarrow{z}}$ to node $\overset{w}{\overrightarrow{az}}$. If we maintain the set F' below, we can detect $LRP(aw)$ in constant time, where

$$F' = \{(\overset{w}{\overrightarrow{x}}, a, \overset{w}{\overrightarrow{ax}}) \mid x, ax \in Factor(w), a \in \Sigma, \text{ and } \overset{w}{\overrightarrow{ax}} = a \cdot \overset{w}{\overrightarrow{x}}\}.$$

Comparing F' and F in Definition 2, one can see that F' is the set of the *labeled reversed suffix links* of $STree'(w)$.

We now have the following theorem.

Theorem 3 *For any $a \in \Sigma$ and $w \in \Sigma^*$, $STree'(w)$ can be updated to $STree'(aw)$ in amortized constant time.*

3.3 Mutual Influences

Here, we consider mutual influences between Left Extension and Right Extension. The next lemma shows what happens to $LRP(w)$ when $STree'(w)$ is updated to $STree'(wa)$.

Lemma 12 *Let $a \in \Sigma$ and $w \in \Sigma^*$. Assume $LRP(w) = LRS(w)$. Let $x = LRS(w)$. If $xa \in Prefix(w)$, then $LRP(wa) = xa$.*

Proof. Since $xa \in Prefix(w)$, $LRS(wa) = xa$. Thus $xa = LRP(wa)$. □

This lemma shows when and where $LRP(wa)$ moves from the location of $LRP(w)$ according to the character a newly added to the right of w. Examining the precondition, "if $xa \in Prefix(w)$", is feasible in $O(|\Sigma|)$ time, which regarded as $O(1)$ if Σ is a fixed alphabet.

The following lemma stands in contrast to Lemma 12.

Lemma 13 *Let $a \in \Sigma$ and $w \in \Sigma^*$. Assume $LRP(w) = LRS(w)$. Let $x = LRP(w)$. If $ax \in Suffix(w)$, then $LRS(aw) = ax$.*

This lemma shows when and where $LRS(aw)$ moves from the location of $LRS(w)$ according to the character a newly added to the left of w. Examining the precondition, "if $ax \in Suffix(w)$", is also feasible in $O(|\Sigma|)$ time, and moving from the location of $LRS(w)$ to that of $LRS(aw)$ can be done in constant time by the use of the labeled reversed suffix link of $LRP(w)$.

As a result of discussion, we finally obtain the following:

Theorem 4 *For any string $w \in \Sigma^*$, $STree'(w)$ can be constructed in bidirectional manner and in $O(|w|)$ time.*

A bidirectional construction of $STree'(w)$ with $w =$ `cocoon` is displayed in Figure 3.

4 Concluding Remarks

We introduced an algorithm for bidirectional construction of suffix trees, which performs in linear time. It should be noted that the proposed algorithm can construct an index of w^{rev} at the same time, where w^{rev} is the reversal of a given string w. In [14], we improved Ukkonen's algorithm so as to construct not only $STree'(w)$ but also $DAWG(w^{\text{rev}})$ in right-to-left on-line manner. The algorithm of this paper leads bidirectional construction of $STree'(w)$ and $DAWG(w^{\text{rev}})$, although theoretical details are omitted in this draft.

Acknowledgment

The author wishes to thank Prof. Ayumi Shinohara and Prof. Masayuki Takeda. Daily fruitful and enthusiastic discussion with them led the author to the inspiration for this work.

References

[1] A. Apostolico. The myriad virtues of subword trees. In A. Apostolico and Z. Galil, editors, *Combinatorial Algorithm on Words*, volume 12 of *NATO Advanced Science Institutes, Series F*, pages 85–96. Springer-Verlag, 1985.

[2] M. Balík. Implementation of dawg. In *Proc. The Prague Stringology Club Workshop '98 (PSCW'98)*. Czech Technical University, 1998.

[3] A. Blumer, J. Blumer, D. Haussler, A. Ehrenfeucht, M. T. Chen, and J. Seiferas. The smallest automaton recognizing the subwords of a text. *Theoretical Computer Science*, 40:31–55, 1985.

[4] A. Blumer, J. Blumer, D. Haussler, R. McConnell, and A. Ehrenfeucht. Complete inverted files for efficient text retrieval and analysis. *J. ACM*, 34(3):578–595, 1987.

[5] D. Breslauer. The suffix tree of a tree and minimizing sequential transducers. *Theoretical Computer Science*, 191:131–144, 1998.

[6] M. T. Chen and J. Seiferas. Efficient and elegant subword tree construction. In A. Apostolico and Z. Galil, editors, *Combinatorial Algorithm on Words*, volume 12 of *NATO Advanced Science Institutes, Series F*, pages 97–107. Springer-Verlag, 1985.

[7] M. Crochemore. Transducers and repetitions. *Theoretical Computer Science*, 45:63–86, 1986.

[8] M. Crochemore and W. Rytter. *Text Algorithms*. Oxford University Press, New York, 1994.

[9] M. Crochemore and R. Vérin. On compact directed acyclic word graphs. In J. Mycielski, G. Rozenberg, and A. Salomaa, editors, *Structures in Logic and Computer Science*, volume 1261 of *Lecture Notes in Computer Science*, pages 192–211. Springer-Verlag, 1997.

[10] M. Farach. Optimal suffix tree construction with large alphabets. In *Proc. The 38th Annual Symposium on Foundations of Computer Science (FOCS '97)*. IEEE Computer Society, 1997.

[11] R. Giegerich and S. Kurtz. From Ukkonen to McCreight and Weiner: A unifying view of linear-time suffix tree construction. *Algorithmica*, 19(3):331–353, 1997.

[12] D. Gusfield. *Algorithms on Strings, Trees, and Sequences*. Cambridge University Press, New York, 1997.

[13] S. Inenaga, H. Hoshino, A. Shinohara, M. Takeda, and S. Arikawa. Construction of the CDAWG for a trie. In *Proc. The Prague Stringology Conference '01 (PSC'01)*. Czech Technical University, 2001.

[14] S. Inenaga, H. Hoshino, A. Shinohara, M. Takeda, and S. Arikawa. On-line construction of symmetric compact directed acyclic word graphs. In *Proc. of 8th International Symposium on String Processing and Information Retrieval (SPIRE'01)*, pages 96–110. IEEE Computer Society, 2001.

[15] S. Inenaga, H. Hoshino, A. Shinohara, M. Takeda, S. Arikawa, G. Mauri, and G. Pavesi. On-line construction of compact directed acyclic word graphs. In A. Amir and G. M. Landau, editors, *Proc. 12th Annual Symposium on Combinatorial Pattern Matching (CPM'01)*, volume 2089 of *Lecture Notes in Computer Science*, pages 169–180. Springer-Verlag, 2001.

[16] S. Inenaga, A. Shinohara, M. Takeda, and S. Arikawa. Compact directed acyclic graphs for a sliding window. In *Proc. of 9th International Symposium on String Processing and Information Retrieval (SPIRE'02)*, Lecture Notes in Computer Science. Springer-Verlag, 2002. (to appear).

[17] M. G. Maaß. Linear bidirectional on-line construction of affix trees. In R. Giancarlo and D. Sankoff, editors, *Proc. 11th Annual Symposium on Combinatorial Pattern Matching (CPM'00)*, volume 1848 of *Lecture Notes in Computer Science*, pages 320–334. Springer-Verlag, 2000.

[18] U. Manber and G. Myers. Suffix arrays: A new method for on-line string searches. *SIAM J. Compt.*, 22(5):935–948, 1993.

[19] E. M. McCreight. A space-economical suffix tree construction algorithm. *J. ACM*, 23(2):262–272, 1976.

[20] J. Stoye. Affixbäume. Master's thesis, Universität Bielefeld, 1995. (in German).

[21] J. Stoye. Affix trees. Technical Report 2000–4, Universität Bielefeld, Technische Fakultät, 2000.

[22] E. Ukkonen. On-line construction of suffix trees. *Algorithmica*, 14(3):249–260, 1995.

[23] P. Weiner. Linear pattern matching algorithms. In *Proc. 14th Annual Symposium on Switching and Automata Theory*, pages 1–11, 1973.

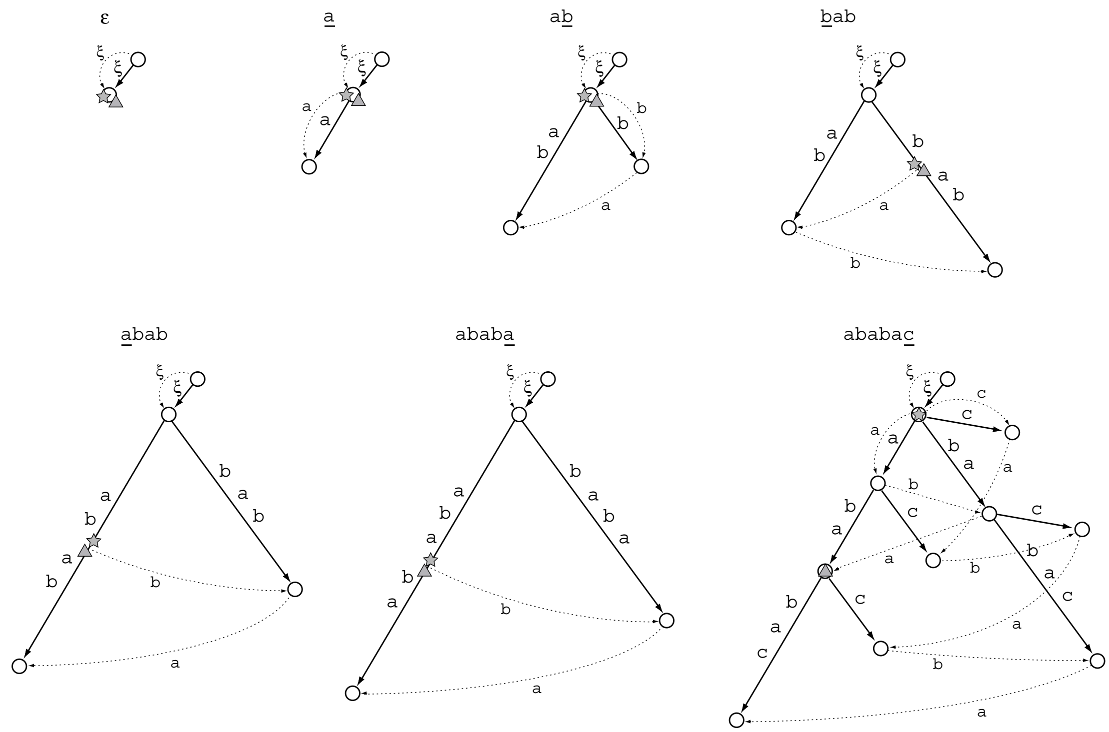

Figure 3: A bidirectional construction of $STree'(w)$ with $w =$ `ababac`. Solid arrows represent edges while dotted arrows denote labeled reversed suffix links. On Right Extension, labeled reversed suffix links are used for the reversed direction, that is, as "normal" suffix links. In each phase, a gray triangle (star, respectively) indicates the location of the longest repeated prefix (suffix, respectively). The newly added character is underlined in each phase. When $STree'$(`ab`) is updated to $STree'$(`bab`), the node for string `b` becomes implicit (Proposition 8). Due to the conversion of $STree'$(`bab`) into $STree'$(`abab`), LRP(`abab`) moves via the labeled reversed suffix link, and LRS(`abab`) also moves to the same position according to Lemma 13. Then, the suffix tree is updated to $STree'$(`ababa`) where LRS(`ababa`) moves while spelling out the new character `a` along the edge. Note that LRP(`ababa`) also moves due to Lemma 12. Since the precondition of Lemma 12 is not satisfied in the string `ababac`, LRP(`ababac`) does not move in $STree'$(`ababac`). For smart construction, we also maintain the labeled reversed suffix link of the longest repeated suffix even if it is *not* on an explicit node (see $STree'$(`bab`), for instance). This labeled reversed suffix link is the only suffix link that would be "modified" after it is created. For example, the labeled reversed suffix link of the node for string `a` in $STree'$(`a`) is deleted in $STree'$(`ab`) since it no longer satisfies the definition of labeled reversed suffix links. On the other hand, that of the node for string `ab` in $STree'$(`abab`) still exists in $STree'$(`ababa`) as that of the node for string `aba`.