Technical Reports on Mathematical and Computing Sciences: TR-C185
title: A New Full-Text Search Algorithm Using Factor Oracle as Index
author: Ryoichi Kato
affiliation:
Dept. of Mathematical and Computing Sciences, Tokyo Institute of Technology
Meguro-ku Ookayama, Tokyo 152-8552, Japan
(`Ryoichi.Kato@is.titech.ac.jp`)

abstract: A new almost linear time algorithm for indexed full-text search is presented. The algorithm uses a modified factor oracle as an index database. A factor oracle is an acyclic automaton made from an input text, and it recognizes all of substrings in the text. It can be built in time proportional to the text length, and it is more space economical and easy to implement comparing similar machineries such as suffix-tree or suffix-automaton. On the other hand, it has a problem that it may accept strings not in the text; thus, we cannot use factor oracles for indexed searching directly. We solve this problem and realize full-text-indexing using a modified factor oracle. The search algorithm runs in time almost proportional to the length of pattern, and it is again easy to implement.

1 Introduction

The string-search problem considered in this paper is for a pattern w and a text p, to detect occurrences of w as a substring of p. If w is a substring of p, we may also want to know the position of one of/all occurrences of w in p. Such string search algorithms can be roughly classified into two types: on-line searching (without building index) and indexed searching. If a text is queried many times, it is worth pre-processing the text and build an index. In this paper, we present a new *indexed* search algorithm using factor oracle [ACR99] as full-text-search index.

Factor oracles are deterministic acyclic automata built from a given text. Factor oracles are more space economical and easy to implement comparing to similar machineries such as suffix-trees [Wei73, McC76, Apo85, Ukk95] or suffix-automata [BBE$^+$85, CV97, IHS$^+$01]. It also runs with less memory than these machineries. However, unlike suffix-trees or suffix-automata, which precisely accept only substrings of the input text, factor oracle may accept strings not in the text. So, factor oracles cannot be used directly as an index for full-text search, while a search algorithm similar to backward-DAWG-match(BDM)[CCG$^+$94] (online search) is proposed in [LL00]. Note that a pattern rejected by any factor oracle is certainly not a substring of the input text from which the factor oracle is built.

The difficulty of using a factor oracle $\mathsf{Oracle}(p)$, a factor oracle built from text p, for index of full-text search algorithm is that, even if pattern w is accepted by $\mathsf{Oracle}(p)$, we are still unsure whether w is a substring of p. In this paper, we propose an algorithm to determine whether pattern w is a substring of text p when w is accepted by $\mathsf{Oracle}(p)$. It opens a way for full-text search using factor oracles as indexes. We used slightly modified factor oracle for this purpose. We also present a method to locate all occurrences of pattern in the text.

A factor oracle $\mathsf{Oracle}(p)$ has $|p|+1$ states and $|p|$ to $2|p|-1$ transitions. It can be built in time proportional to the length of p. Any substring of p can be recognized by the $\mathsf{Oracle}(p)$ When w is accepted by $\mathsf{Oracle}(p)$ at state i, our algorithm can be ran in $\mathcal{O}(|w| \times |\mathsf{SP}^{-1}(i)|)$

time to determine whether w is a substring of p, where $|\mathit{SP}^{-1}(i)|$, number of incoming suffix link, is a nonnegative integer at most $|p| - i$. Although $|\mathit{SP}^{-1}(i)|$ can be $\mathcal{O}(|p|)$ in worst case, we can roughly consider that the value averages about 1, because summation of this value on overall states in $\mathit{Oracle}(p)$ is also $|p|$. Hence we estimate throughout running time of this search algorithm to be almost $\mathcal{O}(|w|)$ in most cases. To determine all position of occurrences of w in p, it takes extra time proportional to the number of occurrences.

2 Factor Oracle

In this section, preliminarily notions, definitions and lemmas upon basic properties of factor oracles are introduced. Most of them appear in [ACR01].

First of all, let us define the notations and the definitions used throughout this document.

- Let $p = p[1 \ldots m]$ be a string of length $|p| = m$ over an arbitrary alphabet Σ.
- Let ϵ be the empty string ($|\epsilon| = 0$).
- Relation $a \overset{\text{Suf}}{\prec} b$ ($\overset{\text{Pre}}{\prec}$ respectively) denotes that a is a suffix (prefix) of b. That is, with appropriate $u \in \Sigma^*$, b can be written $b = ua$ ($b = au$). If $u \neq \epsilon$, we say a is a *proper* suffix (prefix) of b and may be written $a \overset{\text{Suf}}{\precneqq} b$ ($a \overset{\text{Pre}}{\precneqq} b$)
- Relation $a \overset{\text{Fac}}{\prec} b$ denotes a is a substring of b. That is, with appropriate $u, v \in \Sigma^*$, b can be written $b = uav$. If $u, v, a \neq \epsilon$, we say a is a *proper* substring of b and may be written $a \overset{\text{Fac}}{\precneqq} b$
- We denote by $\overline{w}$ the state in automaton reading of string w ends.

Factor oracle is introduced in [ACR01] along with related concepts to it on strings such as *longest repeated suffix* ($\mathit{repet}_p(i)$), *minimal length path* ($\mathit{min}(i)$), etc.

Definition 2.1 ([ACR01]). $\mathit{Oracle}(p)$, defined by its construction algorithm in [ACR01], is an acyclic automaton which recognizes at least all of substrings of p. It has exactly $m + 1$ states(nodes), exactly m internal transitions, and at most $m - 1$ external transitions. Note that all of states are terminating. It also has exactly m additional links called *suffix link.* which are not a part of automaton.

Figure 1 shows Oracle(“abbbaab”).

Regarding factor oracle $\mathit{Oracle}(p)$, “w appears at position i of p” means $w = p[i-|w|+1 \ldots i]$ because we identify an occurrence of a substring w in p by its ending position i.

Internal transitions of factor oracle has one-to-one correspondence with letters in p. In other words, for states i and j ($i < j$), sequence of internal transitions from i to j corresponds to substring $p[i + 1 \ldots j]$. So, we consider position index of p and its corresponding state of $\mathit{Oracle}(p)$ interchangeably. For example, expression “w appears at position i of p” is equivalent to “w appears at state i of $\mathit{Oracle}(p)$”.

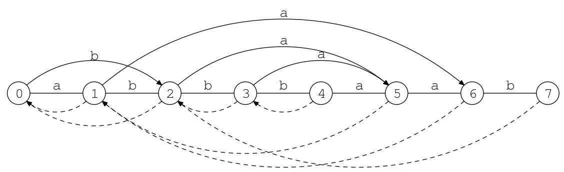

Figure 1: Oracle("abbbaab")
Internal transitions are shown as straight line. Solid arcs left to right on upper side are external transitions. Dashed arcs right to left on downside are suffix links.
Note that internal transitions has one-to-one correspondence with letters of the input text.

- $\mathit{endpos}_p(w) = \{i \mid \exists u \in \Sigma^*, p = uwp[i+1]\dots p[m]\}$
  (Set of ending positions of all occurrence of w as substring in p.)
- $\mathit{poccur}(w, p) = \min\{\mathit{endpos}_p(w)\}$
  (End position of the first occurrence of w in p when $w \overset{\text{Fac}}{\prec} p$)
- $\mathit{min}(i)$ is the minimal length string recognized at state $i \in \mathit{Oracle}(p)$,
- $\mathit{repet}_p(i)$ is the longest suffix of $p[1\dots i]$ which occurs at least twice in $p[1\dots i]$
  (*Longest repeated suffix of* i)

[ACR01] shows many interesting, and important lemmas on factor oracle. Among them, we use the following ones.

Lemma ([ACR01] Lemma 1). Let $u \in \Sigma^*$ be a minimal length string among the strings recognized at state i (i.e., $\mathit{min}(i) = u$) of $\mathit{Oracle}(p)$. Then $u \overset{\text{Fac}}{\prec} p$ and $i = \mathit{poccur}(u, p)$

Lemma ([ACR01] Lemma 2). Let i be a state of $\mathit{Oracle}(p)$ and $u = \mathit{min}(i)$. Then u is a suffix of any string $v \in \Sigma^*$ recognized at state i

Lemma ([ACR01] Lemma 3). Let $w \overset{\text{Fac}}{\prec} p$. w is recognized by $\mathit{Oracle}(p)$ in a state $j \leq \mathit{poccur}(w, p)$

Lemma ([ACR01] Lemma 4). Let i be a state of $\mathit{Oracle}(p)$ and $u = \mathit{min}(i)$. Any path ending by u leads to a state $j \geq i$

The construction algorithm of factor oracle is very similar to that of DAWG (suffix automaton), as remarked in [LL01]. Like DAWG and its family (suffix trie, suffix tree, compact-DAWG), construction algorithm of factor oracle also uses suffix links to perform quick construction. In figure 1, suffix links are showed as dashed arc right to left.

Definition 2.2 ([ACR01]). *suffix link* $S(i)$ of state $i \in \mathit{Oracle}(p)$ is equivalent to $\overline{\mathit{repet}_p(i)}$. We arbitrary set $S(0) = -1$

Lemma ([ACR01] Lemma 7). Let $s > 0$ be a state of $\mathit{Oracle}(p)$ such that $S(s)$ is not 0. Then $\mathit{repet}_p(S(s)) \overset{\text{Suf}}{\prec} \mathit{repet}_p(s)$.

Suffix links are merely an aid of construction and are not components of factor oracle if we interpret factor oracle as plain automaton. Nevertheless, suffix links also play important role in our algorithm.

According to the definition of suffix link, $\mathsf{S}(i) < i$ always holds for arbitrary state $i \in \mathsf{Oracle}(p)$. So we can define a sequence of states for any $i \in \mathsf{Oracle}(p)$, which starts from a state $i = k_0$, ends in state 0 and each state k_j in the sequence is followed by $\mathsf{S}(k_j)$: $k_0 = i, k_1, k_2, \ldots, k_t = 0$ ($k_{j+1} = \mathsf{S}(k_j)$). We call such a sequence of the states "suffix path".

Definition 2.3 ([ACR01]). $\mathsf{SP}(i)$: Suffix path $\mathsf{SP}(i)$ is defined as a set of states on the suffix path starts at state i. i.e., $\mathsf{SP}(i) = \{i, \mathsf{S}(i), \mathsf{S}(\mathsf{S}(i)), \ldots, 0\}$

Lemma ([ACR01] Lemma 13). For any $u \overset{\mathrm{Suf}}{\prec} p$, $\overline{u} \in \mathsf{SP}(p)$.

Definition 2.4. $\mathsf{S}^{-1}(i)$ is a set of all states j of $\mathsf{Oracle}(p)$ such that $\mathsf{S}(j) = i$.

Intuitively, $\mathsf{S}^{-1}(i)$ means tracing all of incoming suffix link in inverse direction.

Corollary 2.1. Since each state $i \in \mathsf{Oracle}(p)$ has exactly one suffix link destination $\mathsf{S}(i) < i$, entire suffix links of a factor oracle can be interpreted as a tree structure. The state 0 is root of such tree and for any node(state) i, its parent is equal to $\mathsf{S}(i)$, and children are equal to $\mathsf{S}^{-1}(i)$ which may be $\emptyset$. We call such a tree suffix-link-tree.

Definition 2.5. $\mathsf{SP}^{-1}(i)$ is a set of all of descendant nodes of i in the suffix-link-tree of $\mathsf{Oracle}(p)$ i.e., $\mathsf{SP}^{-1}(i) = \{j | i \in \mathsf{SP}(j)\}$

In other words, $\mathsf{SP}^{-1}(i)$ is set of all states in a subtree of suffix-link-tree where i is the root of such subtree.

Beside suffix links, minimal length path is another key to reach our main result.

Lemma 2.1. Let $i > 1$ be state of $\mathsf{Oracle}(p)$. Then $|\mathsf{min}(\mathsf{S}(i))| < |\mathsf{min}(i)|$.

Proof. $i = 1$. Verified by $|\mathsf{min}(i) = 1|$, $|\mathsf{min}(\mathsf{S}(i))| = |\mathsf{min}(0)| = 0$.

$\mathsf{S}(i) = 0$. Verified by $\mathsf{min}(0) = 0$, $|\mathsf{min}(i)| \geq 1$.

$\mathsf{S}(i) > 0$. By induction. Assume every state s ($0 \leq s \leq i$), satisfies $|\mathsf{min}(S(s))| < |\mathsf{min}(s)|$. Let $j = \mathsf{S}(i)$. Let $k \in \mathsf{SP}(i-1)$ to be a state such that $j = \delta(k, \sigma)$ (By $j \neq 0$, existence of such k is assured).

Consider set L of all states which leads to state i with one transition. $L = \{l' | \delta(l', \sigma) = i\}$. Then following relation holds.

$$|\mathsf{min}(i)| = \min\left\{|\mathsf{min}(l')| \,\middle|\, l' \in L\right\} + 1$$

From the definition of construction algorithm, $L \subset \mathsf{SP}(i-1)$. Let l be the minimal(leftmost) state among states in L. Then l and i satisfies relation $|\mathsf{min}(i)| = |\mathsf{min}(l)| + 1$ by induction hypothesis.

(Figure 2 illustrates this)

$$
\begin{aligned}
|\mathit{min}(i)| &= |\mathit{min}(l)| + 1 \\
&\geq |\mathit{min}(k)| + 2 && \text{— (1) }^{*1} \\
|\mathit{min}(k)| + 1 &\geq |\mathit{min}(j)| && \text{— (2) }^{*2}
\end{aligned}
$$

$$
\begin{aligned}
|\mathit{min}(i)| &\geq |\mathit{min}(k)| + 2 && \text{— from (1)} \\
&> |\mathit{min}(j)| && \text{— from (2)}
\end{aligned}
$$

^{*1} $S(l) = k$ (definition of construction algorithm), induction hypothesis.

^{*2} $\delta(k, \sigma) = j$

□

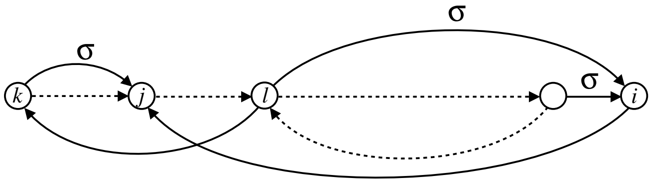

Figure 2: relations among i, $i-1$, $j = S(i)$
Arrows left from right are transitions. Right from left are suffix links. Solid arrows represent one transition or suffix link. Dash arrows represent multiple of transitions or suffix links.

Lemma 2.2. For any state $i \in \mathit{Oracle}(p)$, let $j = S(i)$. Then $\mathit{min}(j) \overset{\text{Suf}}{\prec} \mathit{min}(i)$. [1]

Proof. By the definition of the construction algorithm, $\overline{\mathit{repet}_p(i)} = S(i) = j$. [ACR01]–Lemma 2 and 7 assures $\mathit{min}(j) \overset{\text{Suf}}{\preceq} \mathit{repet}_p(i)$.

Assume $\mathit{min}(i) \overset{\text{Suf}}{\preceq} \mathit{repet}_p(i)$, then $\mathit{poccur}(\mathit{min}(i), p) \leq \mathit{poccur}(\mathit{repet}_p(i), p) < i$ from [ACR01]–Lemma 3 because $\mathit{poccur}(\mathit{repet}_p(i), p) < i$ by the definition of $\mathit{repet}_p(i)$. This contradicts [ACR01]–Lemma 1. Hence $\mathit{repet}_p(i) \overset{\text{Suf}}{\prec} \mathit{min}(i)$, and with the first half of this proof, $\mathit{min}(j) \overset{\text{Suf}}{\prec} \mathit{min}(i)$ is clarified. □

For every states $i \in \mathit{Oracle}(p)$, $\mathit{min}(i)$ can be calculated Using NewAddLetter instead of AddLetter of [ACR01]. NewAddLetter exploits property of $\mathit{min}(i)$ appeared in the proof of Lemma 2.1

Algorithm 1 NewAddLeter($\mathit{Oracle}(p)$, σ)
1. Create a new state $m+1$
2. Create a new transition from m to $m+1$ by σ

[1] Lemma:2.1 is trivialized by this lemma.

```
 3.  minlen[m+1] ← minlen[m]+1                               /* Added */
 4.  k ← S(m)
 5.  while ( k > −1 ∧ No transition from k by σ)
 6.      Create a new transition from k to m by σ
 7.      minlen[m + 1] ← minlen[k]+1                         /* Added */
 8.      k ← S(k)
 9.  if (k = −1)
10.      S(m + 1) ← 0
11.  else
12.      S(m + 1) ← δ(k, σ)
```

Lemma 2.3. By using algorithm NewAddLeteer instead of AddLetter for Build_Oracle in [ACR01], $|\mathit{min}(i)|$ for every state $i \in \mathit{Oracle}(p)$ can be calculated without loosing linearity of the original construction algorithm.

Proof. Only lines 3 and 7 is added to the original routine and both of them can be done in $\mathcal{O}(1)$. □

3 Finding Position of Substring Occurrence

Consider the case that a pattern w is accepted by $\mathit{Oracle}(p)$. If we are lucky, $p\big[(\overline{w} - |w| + 1) \ldots \overline{w}\big] = w$ when w is accepted by $\mathit{Oracle}(p)$, and in this case, $w \overset{\text{Fac}}{\prec} p$ and $\mathit{poccur}(w, p) = \overline{w}$. Difficulty arises if w is accepted by $\mathit{Oracle}(p)$ but $p\big[(\overline{w} - |w| + 1) \ldots \overline{w}\big] \neq w$. In this case, w may be not a substring of p or may appears at somewhere else other than $\overline{w}$.

In this section, we present an algorithm to determine whether w is a substring of p or not when a pattern w accepted by $\mathit{Oracle}(p)$. The algorithm we present is very simple. It can be determined by only checking if w appears at any of states in set $S^{-1}(\overline{w})$. More intuitively, looking only all the nodes directly under the accepting state of w in the suffix-link-tree is enough to solve this problem. This can be done in time $\mathcal{O}(|w| \times S^{-1}(\overline{w}))$.

Moreover, if $w \overset{\text{Fac}}{\prec} p$, we can easily locate all position of occurence of w in p ($\mathit{endpos}_p(w)$) with extra $\mathcal{O}(|\mathit{endpos}_p(w)|)$ time.

3.1 Suffix-Link-Tree and Occurence of Substring

To begin with, let us elucidate relation between substring occurence and suffix-link-subtree.

Lemma 3.1. For any $w \overset{\text{Fac}}{\prec} p$, $\mathit{endpos}_p(w) \subset SP^{-1}(\overline{w})$

Proof. From [ACR01]–Lemma 13, $\overline{w} \in SP(i)$ for all $i \in \mathit{endpos}_p(w)$

From [ACR01]–Lemma 3, $\forall i \in \mathit{endpos}_p(w), \overline{w} \leq i$.

Hence all states $i \in \mathit{endpos}_p(w)$ are descendant of $\overline{w}$ in suffix-link-tree of $\mathit{Oracle}(p)$. In other words, all elements of $\mathit{endpos}_p(w)$ are in the suffix-link-subtree of state $\overline{w}$. □

The lemma assures that if w accepted by $\mathit{Oracle}(p)$, all of occurrence of w in p appear, if any, as only descendant of suffix-link-subtree $SP^{-1}(\overline{w})$. Besides, following corollary can be led by Lemma2.2.

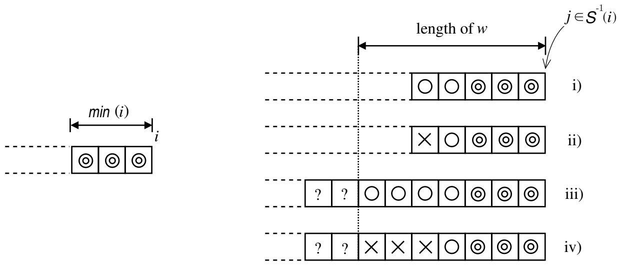

Figure 3: relation between i, j and $\mathit{min}(i)$
One square represents one internal transition (equivalent to one character) in factor oracle. Boundary between squares correspond to a state. Squares shown explicitly represents $\mathit{min}(i)$(left) and $\mathit{min}(j)$(right) and $\bigcirc$, $\times$ in it are characters which is accord and not accord with w respectively.
$\circledcirc$ represents suffix which guaranteed to be accord with suffix of w by [ACR01]–Lemma 2, and Lemma2.2.
i) can not happen. (Lemma3.2)

Corollary 3.1. For any state $i \in \mathit{Oracle}(p)$, if $i \in \mathit{endpos}_p(w)$ and $w \overset{\text{Suf}}{\preceq} \mathit{min}(i)$, then $\mathit{SP}^{-1}(i) \subset \mathit{endpos}_p(w)$. On the other hand, if $i \notin \mathit{endpos}_p(w)$ and both $\mathit{min}(i) \overset{\text{Suf}}{\not\preceq} w$ and $w \overset{\text{Suf}}{\not\preceq} \mathit{min}(i)$, [2] $\mathit{SP}^{-1}(i) \cap \mathit{endpos}_p(w) = \emptyset$.

From above lemma and corollary, we can deduce the following.

Lemma 3.2. Let w be a substring of p which accepted by $\mathit{Oracle}(p)$ at state $i = \overline{w}$.

Then, for any $j \in S^{-1}(i)$, relations between j and w can be split into below four cases if $S^{-1}(i) \neq \emptyset$.

i) $|\mathit{min}(j)| \lneq |w|, \quad \mathit{min}(j) \overset{\text{Suf}}{\preceq} w$

ii) $|\mathit{min}(j)| \lneq |w|, \quad \mathit{min}(j) \overset{\text{Suf}}{\not\preceq} w$

iii) $|w| \leq |\mathit{min}(j)|, \quad w \overset{\text{Suf}}{\prec} \mathit{min}(j)$p

iv) $|w| \leq |\mathit{min}(j)|, \quad w \overset{\text{Suf}}{\not\preceq} \mathit{min}(j)$

And we can claim the following.

- case i) never happens.
- $\mathit{SP}^{-1}(j) \cap \mathit{endpos}_p(w) = \emptyset$ in case ii), iv).
- $\mathit{SP}^{-1}(j) \subset \mathit{endpos}_p(w)$ in case iii).

[2]i.e., if the shorter of $\mathit{min}(i)$ and w isn't suffix of the other.

Proof. Since $j \notin \mathit{endpos}_p(w)$ in case of ii), iv) and $j \in \mathit{endpos}_p(w)$ in case of iii), corollary 3.1 clarifies $SP^{-1}(j) \cap \mathit{endpos}_p(w) = \emptyset$ for case ii), iv) and $SP^{-1}(j) \subset \mathit{endpos}_p(w)$ for case iii).

Considering case i), [ACR01]-Lemma 4 requires $i \geq j$ because $\mathit{min}(j) \overset{\text{Suf}}{\prec} w$. This contradicts $\overline{w} < j$. ☐

By the above lemma, now we can determine whether w is a substring of p when w accepted by $\mathit{Oracle}(p)$ with following procedure.

As first step, check if $\overline{w}$ is equal to $\mathit{poccur}(w,p)$ i.e., compare $p[\overline{w} - |w| + 1 \ldots \overline{w}]$ and w. If first step faild, then as secound step, compare w with $p[j - |w| + 1 \ldots j]$ for all states $j \in S^{-1}(\overline{w})$. Lemma 3.2 and Lemma 3.1 assures w is substring of p if and only if either $\overline{w}$ is equal to $\mathit{poccur}(w,p)$ or w appears at at least one of the states $j \in S^{-1}(\overline{w})$. ($p[j - |w| + 1 \ldots j] = w$; case iii) of 3.2).

More over, the lemma claims that the set $\mathit{endpos}_p(w)$ is equal to set of all node in all suffix-link-subtrees of j such that the pattern w appears at j. ($\{\overline{w}\}$ should also be included in the set if $\overline{w} = \mathit{poccur}(w,p)$.) This can be written down as follows:

Theorem 1. Set of all ending position of w's occurrence in p, $\mathit{endpos}_p(w)$ can be described as follows.

$$\mathit{endpos}_p(w) = \begin{cases} \{\overline{w}\} \cup \{k \in SP^{-1}(j) \,\big|\, \text{for all } j \in S^{-1}(\overline{w}) \text{ s.t. } w \text{ appears at } j\ \} \\ \qquad \text{if } \overline{w} = \mathit{poccur}(w,p) \\ \\ \{k \in SP^{-1}(j) \,\big|\, \text{for all } j \in S^{-1}(\overline{w}) \text{ s.t. } w \text{ appears at } j\ \} \\ \qquad \text{otherwise} \end{cases}$$

3.2 Search Algorithm

Preceding results leads us to a straight forward embodiment of string search algorithm using factor oracle as full-text index.

To answer yes / no for existence of w in p, following algorithm should be sufficient.

Algorithm 2 IsSubstring?($\mathit{Oracle}(p)$, w)

```
if w recognized by Oracle(p)
    i ← w̄ of Oracle(p)
else
    return FALSE
if p[(i − |w| + 1) . . . i] = w
    return TRUE                         /* w̄=poccur(w,p) */
else
    foreach j ∈ S^{-1}(i)
        if p[(j − |w| + 1) . . . j] = w
            return TRUE                 /* case iii) of Lemma 3.2 */
        else
            next                        /* case ii) or iv) of Lemma 3.2 */
    return FALSE
```

Theorem 2. Algorithm IsSubstring? can run in in $\mathcal{O}(2|w|+|w|\times|S^{-1}(i)|)$ time to determine whether w is a substring of p.

Proof. In line 1 of algorithm, It takes $\mathcal{O}(|w|)$ time to read w by $\mathit{Oracle}(p)$ since $\mathit{Oracle}(p)$ is an automaton. Line 5 compares $|w|$ of characters, takes another $\mathcal{O}(|w|)$ time. Since 'foreach' block starting at line 8 repeats at most $|S^{-1}(i)|$ times and comparison of line 9 runs in $\mathcal{O}(()|S^{-1}(i)|\times|w|)$ time. □

It is obvious that we can easily modify algorithm IsSubstring? to FindAllOccurence so that the algorithm can locate all w occurences when $w \overset{\text{Fac}}{\prec} p$.

Algorithm 3 FindAllOccurence($\mathit{Oracle}(p)$, w)

1. **if** w recognized by $\mathit{Oracle}(p)$ at state $\overline{w}$
2. $i \leftarrow \overline{w}$ of $\mathit{Oracle}(p)$
3. **else**
4.   **return** $\emptyset$ /* empty set */
5. **if** $p\big[(i-|w|+1)\ldots i\big] = w$
6. $\mathit{endpos}_p(w) \leftarrow i$
7. **foreach** $j \in S^{-1}(i)$
8.   **if** $p\big[(j-|w|+1)\ldots j\big] = w$
9. $\mathit{endpos}_p(w) \leftarrow \mathit{endpos}_p(w) \cup \mathit{SP}^{-1}(j)$
10.   **else**
11.   **next**
12. **return** $\mathit{endpos}_p(w)$

Theorem 3. Execution of algorithm FindAllOccurence needs extra time proportional to $|\mathit{endpos}_p(w)|$ (i.e., number of occurence of w in p) beside running time of IsSubstring?.

Proof. Only line 9 is significant. By theorem 1, there is exactly $|\mathit{endpos}_p(w)|$ (when $w \notin \mathit{endpos}_p(w)$) or $|\mathit{endpos}_p(w)|-1$ (when $w \in \mathit{endpos}_p(w)$) total number of nodes in suffix-link-subtrees traversed on line 9. □

In lines 5 and 8 of IsSubString? and lines 5 and 9 of FindAllOccurence, w is compared against $p\big[(i-|w|+1)\ldots w\big]$ and $p\big[(j-|w|+1)\ldots j\big]$ respectively. But this comparison is redundant if $\mathit{min}(\overline{w})$ is known beforehand. (Remember $\mathit{min}(i)$ for $i \in \mathit{Oracle}(p)$ can be computed by NewAddLetter with slight overhead.) since the fact that w accepted by $\mathit{Oracle}(p)$ at $\overline{w}$ guarantees:

$$\begin{aligned} w\big[(|w|-|\mathit{min}(i)|+1)\ldots|w|\big] &= \mathit{min}(i) && {}^{*1}\\ &= p\big[(j-|\mathit{min}(\overline{w})|+1)\ldots j\big] && {}^{*2}\end{aligned}$$

*1 [ACR01]–Lemma 2

*2 Lemma 2.2

So comparison of $p\big[(i-|w|+1)\ldots(i-|\mathit{min}(i)|)\big]$ or $p\big[(j-|w|+1)\ldots(j-|\mathit{min}(i)|)\big]$ resp. against w is sufficient. This will save $\mathcal{O}(|\mathit{min}(\overline{w})|\times|S^{-1}(\overline{w})|)$ of running time.

4 Preliminary Experiment on Number of Incoming Suffix Links

Running time of the search algorithm depends on $|S^{-1}(\overline{w})|$ for pattern w. Since there can be at most $|p|$ incoming suffix links for a state, running time can be $\mathcal{O}(|w||p|)$ in worst case. However, it not likely be so worse in most cases, because total number of (incoming) suffix links in factor oracle is also $|p|$. In other words, (incoming end of) $|p|$ suffix links are distributed among $|p|$ states. More over, it seems that the longer pattern be, the less incoming links for the accepting state of the pattern tend to be. Intuitively, it is because longer patterns tends to appear less frequently in the text. Following preliminary experimental results indicates $|S^{-1}(\overline{w})|$ is farely small in average case.

We ran search program for factor oracles built from six different text with five different pattern length: 800, 1600, 2400, 3200 and 4000 bytes. Each pattern is extracted randomly from the search target text (So, search never fails). We repeated random extraction of pattern and search 4096 times for each length and counted number of incoming suffix links ($|S^{-1}(\overline{w})|$) of accepting state of the pattern. Table 1 shows averaged result of 4096 random patterns.

file	size(MB)	800	1600	2400	3200	4000
E.coli	4.64	1.06	1.11	1.12	0.91	0.86
bible.txt	4.05	0.98	0.98	0.94	0.92	1.01
chromo.seq	7.18	0.94	0.96	0.90	0.93	1.03
emacs21-x	4.30	0.04	0.00	0.00	0.00	0.00
PostScript	6.13	7.03	1.95	1.44	1.37	1.26
world192.txt	2.47	1.05	0.99	0.94	1.00	1.08

Table 1: average $|S^{-1}(\overline{w})|$ for random 4096 subsequences as pattern

Average number of incoming suffix links of accepting state of pattern, length 800, 1600, 2400, 3200 and 4000. E.coli, chromo.seq are DNA sequences, emacs21-x is linux-386-x11 binary of gnu-emacs version 21.3. world192.txt is "The World Factbook 1992".

Not only number of incoming suffixlinks, but how many of them are from the states such that the pattern appears is also significant to the execution speed of the search algorithm. When the pattern appears at majority of states among $S^{-1}(\overline{w})$, it is likely that algorithm stops in early stage of 'foreach' loop in algorithm IsSubstring? line 8. Table 1 shows percentage of such states against $|S^{-1}(\overline{w})|$ as 'hit rate' over same set of texts and patterns as table 1.

5 Conclusion

A new full-text indexed search algorithm using factor oracle is presented.

Although acception of a pattern string in a factor oracle not necessarily mean that the pattern is a substring of the text on which the factor oracle was built, with the algorithm presented, we can easily determine wether the pattern is a substring of the text or not. It can be done by only looking through all of states such that has suffix link to the acception state of the pattern. To make it possible, we did slight modification to the structure of factor oracle; We introduced 'inverse suffix link', which is merely a suffix link in inverse direction. The

file	hit rate
E.coli	58.77%
bible.txt	68.61%
chromo.seq	58.46%
emacs21-x	84.60%
PostScript	76.16%
world192.txt	70.91%

Table 2: average hit rate

search algorithm runs in time proportional to length of the pattern multiplied by the number of incoming suffix link to the acception state. More over, it is also easy to locate all position of occurrence of the pattern in text with extra time proportional to the number of occurrence of pattern.

Although the number of incoming suffix links of the acception state of a pattern may range from 0 to the size of the search target text, since there is exactly same number as states of suffix links through out whole factor oracle, the number of incoming suffix links can roughly considered to be about 1 for a state in averege. Preliminary experiment also indicates that the number of suffix links to a state is farely small for most cases and it can be said that running time of the algorithm is virtually linear to the size of a pattern.

References

[ACR99] Cyril Allauzen, Maxime Crochemore, and Mathieu Raffinot. Factor oracle: A new structure for pattern matching. In *Conference on Current Trends in Theory and Practice of Informatics*, pages 295–310, 1999.

[ACR01] Cyril Allauzen, Maxime Crochemore, and Mathieu Raffinot. Efficient experimental string matching by weak factor recognition. *Lecture Notes in Computer Science*, 2089:51–??, 2001.

[Apo85] A. Apostolico. The myriad virtues of suffix trees. In *In Combinatorial Algorithms on Words, NATO ISI Series*, pages 85–96. Springer-Verlag, 1985.

[BBE+85] A. Blumer, J. Blumer, A. Ehrenfeucht, D. Haussler, M.T. Chen, and J. Selferas. The smallest automaton recognizing the subwords of a text. In *Theoretical Computer Science*, volume 40, pages 31–55, 1985.

[CCG+94] M Crochemore, A. Czumaj, L. Gasieniec, S. Jarominek, T. Lecroq, W. Plandowski, and W. Rytter. Speeding up two string-matching algorithms. *Algorithmica*, 4(12):247–267, 1994.

[CV97] M. Crochemore and R. Vérin. Direct construction of compact directed acyclic word graphs. In A. Apostolico and J. Hein, editors, *Proceedings of the 8th Annual Sympo-*

sium on Combinatorial Pattern Matching, 1264, pages 116–129, Aarhus, Denmark, 1997. Springer-Verlag, Berlin.

[IHS+01] Shunsuke Inenaga, Hiromasa Hoshino, Ayumi Shinohara, Masayuki Takeda, Setsuo Arikawa, Giancarlo Mauri, and Giulio Pavesi. On-line construction of compact directed acyclic word graphs. In *Combinatorial Pattern Matching*, pages 169–180, 2001.

[LL00] A. Lefebvre and T. Lecroq. Computing repeated factors with a factor oracle. In L. Brankovic and J. Ryan, editors, *Proceedings of the 11th Australasian Workshop On Combinatorial Algorithms*, pages 145–158, Hunter Valley, Australia, 2000.

[LL01] A. Lefebvre and T. Lecroq. Compror: on-line lossless data compression with a factor oracle. In *Information Processing Letters*, volume 83(1), pages 1–6, 2001.

[McC76] Edward M. McCreight. A space-economical suffix tree construction algorithm. *Journal of the ACM*, 23:262–272, 1976.

[Ukk95] E. Ukkonen. On-line construction of suffix trees. *Algorithmica*, 3(14):249–260, 1995.

[Wei73] Peter Weiner. Linear pattern matching algorithms. In *IEEE Symposium on Foundations of Computer Science*, pages 1–11, 1973.