# LONGEST COMMON FACTOR OF TWO WORDS.

Maxime CROCHEMORE

LITP - ROUEN
et
CSP - Université de PARIS-NORD
Avenue J.B Clément
F - 93430 VILLETANEUSE

**Abstract** : The LCF of two words u and v is the maximal length of a common factor of u and v. A linear time algorithm to compute LCF is given, based on a linear time algorithm to build the minimal suffix automaton of a word. The algorithm naturally turns into a real-time string-matching algorithm.

Inside each string-matching algorithm is involved a notion of distance letween words. For instance c lassicalk string-matching algorithms often use a distance related to the longest common prefix of the two strings [KMP 77]. In this paper we show that the use of the f-distance [Ch 83] which is defined from the maximal length of common factors (LCF) of the two strings yields to more powerful algorithms, say, to real-time string-matching algorithms, provided the alphabet is fixed.

The basic structure that is used to deal with the above questions is the minimal deterministic automaton which accepts as language all the suffixes of a word. The size of such on automaton is linear in the length of the word as it was first noted by Blumer and al.. The construction of a suffix automaton also takes a linear time. Suffix automata can be complemented to transducers that produce as output the position of the recognized word. The first two sections contain the results on suffix transducers that are needed to compute the LCF of two words. The material of these sections comes essentially from [BBEHCS 85] and [Cr 86].

Suffix transducers (or factor transducers) are an extension of position trees or suffix trees considered, among others, by Weiner, McCreight and Slisenko [Sl 83]. However the construction given in the second section is in the same vein as those of [KMP 77]. Suffix transducers or related structures have a wide range of applications among which are : string-matching problems including inverted file questions, data compression, decipherability, search for repetitions. In each case linear and optimal algorithms are obtained.

The two last sections of this paper are concerned with the computation of LCF and a transformation of the algorithm which leads to a real-time string-matching algorithm.

## MINIMAL SUFFIX TRANSDUCERS.

Words that are considered here are elements of the free monoid $A^*$ which empty word is denoted by 1. The free semi-group $A^*-\{1\}$ is denoted by $A^+$.

Given a word x in $A^*$, its set of factors is

$$F(x)=\{u\in A^* / \exists y,z\in A^* \ x=yuz\}$$

and its set of suffixes (or right factors) is

$$S(x) = \{u\in A^* / \exists y\in A^* \ x=yu\}.$$

Being finite S(x) can be recognized by a finite state automaton. We only consider incomplete deterministic automata i.e. deterministic automata without sink state. Then, not all the transitions by letters of the alphabet are defined on a given state. Let $\mathcal{S}(x)$ be the minimal such automaton recognizing S(x) :

$$\mathcal{S}(x)=(Q_x, i, T_x, F_x),$$

where $Q_x$ is the set of states, $T_x$ is the set of terminal states, $F_x$ is the set of transitions and i is the initial state. Reading a letter $a\in A$ from a state $q\in Q_x$ leads to a state q', noted q.a, if $(q,a,q')\in F_x$. If $u\in A^* q.u$ is the state reached, if any, after u has been read from state q.

One of the most important properties satisfied by minimal suffix automata $\mathcal{S}(w)$ is the fact that their size is linear in the length of x, $|x|$. Exact bounds are known together with those words that reach the upper bounds [CM 86].

**PROPOSITION 1**: The set $Q_x$ of states of the minimal suffix automaton $\mathcal{S}(x)$ satisfies :
if $|x| \leqslant 2$ then $|Q_x| = |x|+1$,
if $|x| > 2$ then $|x|+1 \leqslant |Q_x| \leqslant 2|x|-1$.
Furthermore, in the second case :

$$|Q_x|=2|x|-1 \text{ iff } x\in ab^*$$

where a and b are two distinct letters of A.

**PROPOSITION 2.** The set $F_x$ of transitions of the minimal suffix automaton $\mathcal{S}(x)$ satisfies :
if $|x| \leqslant 3$ then $|x| \leqslant |F_x| \leqslant 2|x|-1$,
if $|x| > 3$ then $|x| \leqslant |F_x| \leqslant |F_x| \leqslant 3|x|-4$.
Furthermore, in the second case :
$|F_x|=3|x|-4$ iff $x\in ab^*c$
where a,b and c are three distinct letters of A.

Automata $\mathcal{S}(x)$ can be transformed in transducers which output positions of the input factor of x. First, let p be a function defined on F(x) by :

$$p(u) = \min \{ |y| / \exists z \in A^* \; x = yz \text{ et } u \in F(y) \}.$$

The function p is compatible with the right syntactic congruence associated with S(x), which means that, if u and v are factors of x, i.u=i.v implies p(u)=p(v). The function p can thus be defined on states of $\mathcal{S}(x)$ and we get a first transducer where the output associated with a word u is linked to the state i.u.

Another more interesting way to get a transducer is to consider the function pos on F(x) :

$$pos(u) = p(u) - |u|.$$

This function is still a sequential function [Be 79]. As p(u) only depends on i.u, with each transition $(q,a,q') \in F_x$ is associated the output

$$q*a = p(q') - p(q) - 1.$$

Reading a word $u = a_1 a_2 \ldots a_j$ in $\mathcal{S}(x)$ from its initial state i produces the total output :

$$i*a_1 + (i.a_1)*a_2 + \ldots + (i.a_1 a_2 \ldots a_{j-1})*a_j.$$

When u is a suffix of x its position is $|xu^{-1}|$ which is also

$$pos(u) + |x| - p(u).$$

$\mathcal{S}(x)$ becomes a subsequential transducer [Be 79] if, in addition to outputs on transitions, is defined on terminal states q of $\mathcal{S}(x)$ :

$$out\ (q) = |x| - p(q).$$

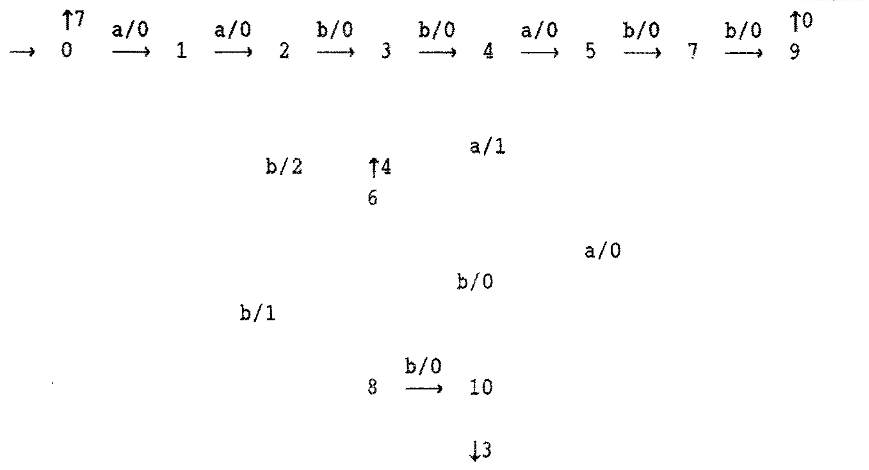

Figure 1. Minimal suffix transducer for aabbabb (terminal states are 0,6,9 and 10).

**CONSTRUCTION**

The second important point concerning minimal suffix automata $\mathcal{S}(x)$ is that their construction can be achieved in time linear in the length of word x. Figure 2 contains an algorithm that builds $\mathcal{S}(x)$. The first 'while' loop is an on-line construction of $\mathcal{S}(x)$ except for the terminal states which are marked with the function 'out' during the last 'while' loop. Note that one or two states are created during each pass through instructions of the first 'while' loop.

The main point in the algorithm is the use of a function s defined on states of $\mathcal{S}(x)$ and which is called suffix link. Its role is analogue to the 'failure' function of the Knuth, Morris and Pratt's string-matching algorithm [KMP 77].

Function s is first defined on non empty factors u of x by :

$$s(u)=\text{longest suffix } w \text{ of } u \text{ such that } i.u\neq i.w.$$

This means that s(u) is the longest suffix of u which appears in a different right context inside x.

**PROPOSITION 3.** Function s is compatible with the right syntactic congruence associated with S(x), i.e. :

$$\forall u,v\in F(x)-\{1\} \ i.u=i.v \Rightarrow s(u)=s(v).$$

Proposition 3 shows that s can be defined on states of $\mathcal{S}(x)$ except on the initial state i. From that point of view, s behaves like a default state function for $\mathcal{S}(x)$.

During execution of the algorithm in figure 2, each time a new state q is created (except at initialization) the value of s(q) is computed with the help of function 'suffix'. Its writing has been made simpler by introducing an artificial state on which are defined transitions to the initial state i by all the letters occurring in x. The function 'suffix', called with parameters r and a, climbs up the suffix links starting from r until is encountered a state on which a transition by a is defined.

Using side effect, function 'suffix' creates or updates transitions.

Tests in the algorithm and function 'suffix' are done by looking at the value of the function l defined on states q of $\mathcal{S}(x)$ :

$$l(q)=\max\{|w|/w\in A^* \text{ and } i.w=q\}.$$

One of the properties that brings all its simplicity to the algorithm is the fact that suffix links on words are of maximal length inside their right syntactic congruence class. This partly explains why function 'suffix' does not have to create states.

**PROPOSITION 4.** Let $y\in F(x)-\{1\}$ and q be that state $\mathcal{S}(x)$ such that i.y=q. Let u=s(y). Then

$$\forall v\in F(x) \ i.v=i.u \Rightarrow |v| \leqslant |u|,$$

which may be translated as

$$|u| = l(s(q)).$$

On a given finite alphabet, all instructions of the algorithm in figure 2 take a constant time, except calls to 'suffix'. To conclude on the $O(|x|)$ global time complexity, observe that that each recursive call to suffix strictly decreases $l(s(last))$ which is increased by at most 1 unit for each new letter.

**Theorem 5.** The construction of the minimal suffix transducer $\mathcal{S}(x)$ by the algorithm in figure 2 is achieved in time $O(|x|)$.

Almost the same results as those related in this section and the preceeding one remain true when one deals with factor automata instead of suffix automata. But minimal factor automata are of no use to compute LCF because they do not satisfy proposition 5.

```
begin create state 'art'; l(art)←p(art)←-1 ;
   creat state i ; l(i)←p(i)←0 ; s(i)←art ;
   last ← i ;
   while not end of input do
      read next letter a ;      art.a←i ;
      create state q ;
      l(q)←l(last)+1;  p(q)←p(last)+1 ;
      last.a←q ; last*a← 0 ;
      r←suffix (last,a) ;
      if l(r.a) > l(r)+1 then
         create state $\bar{r}$ ; with same transitions as r.a ;
         l($\bar{r}$) ← l(r)+1 ; p($\bar{r}$)←p(r.a) ; s(r.a)←$\bar{r}$ ;
         r.a ← $\bar{r}$ ; r*a ← p($\bar{r}$)-p(r)-1 ;
         s($\bar{r}$)←suffix (r,a).a ;
      end if ;
      s(q)←r.a ; last ←q ;
   end while ;
   q←last ; out (q)←0 ;
   while  q≠i do
      q←s(q) ; out (q)←p(last)-p(q) ;
   end while ;
end.


function suffix (r,a) ;
   if s(r).a not defined or l(s(r).a)≥l(r.a) then
      s(r).a←r.a ; s(r)*a←p(s(r).a)-p(s(r))-1 ;
      return (suffix (s(r),a)) ;
   else return (s(r)) ;
   end if ;
end function.
```

**Figure 2 - Construction of minimal suffix tranducers.**

## F - DISTANCE AND LCF.

This section deals with string-matching questions of the kind : search a text t for an occurrence of a word x. The suffix transducer of text t brings an interesting solution to this problem since any further search for a word x takes a time $O(|x|)$.

We concentrate on another solution which is more convenient when text t often changes as it is the case under a text editor. This time the suffix transducer $\mathcal{S}(x)$ is used in a particular way by means of its suffix link s and the function l which gives, for a state q in $\mathcal{S}(x)$, the length of a longest word that reaches q from i.

Given $x,t \in A^*$, we introduce the function

$$LCF(x,t)=\max\{|w|/w \in F(x) \text{ and } w \in F(t)\}.$$

From LCF is defined a distance d between words of $A^*$, called the f-distance [Ch 83] :

$$d(x,t)=|x|+|t|-2LCF(x,t).$$

Searching t for an occurrence of x translates to searching for a factor u of t such that $|u|=|t|$ and $LCF(x,u)=|x|$ or $d(x,u)=0$.

The algorithm in figure 3 is the basic algorithm to compute LCF(x,t) or d(x,t). It may readily be adapted to do string-matching or even approximative string-matching. The algorithm uses the suffix automaton $\mathcal{S}(x)$ already built. So, $\mathcal{S}(x)$ can be considered as one of the inputs of the algorithm. The other input is the text t.
If $t=t_1t_2...t_n$, where the $t_i$'s belongs to A, the output of algorithm in figure 3 is the sequence $l_0,l_1,...,l_n$ defined by

$$l_k=\max\{|w|/w \in F(x) \text{ and } w \in S(t_1...t_k)\}.$$

With this notation we get

$$LCF(x,t)=\max\{l_k/k=0,...,n\}.$$

The proof that the algorithm works well lies on proposition 5 which contains a property of function l on states that are images by the suffix link s.

To see why the time complexity of the algorithm is globally $O(|t|)$, it is enough to note that the instruction '$q \leftarrow s(q)$' of the internal 'while' loop strictly decreases l(q) from its value $l_k$, and besides, this latter quantity increases by at most 1 for each letter of t.

**Theorem 6.** Algorithm in figure 3 compute the lengthes of the common factors of x and t in time $O(|t|)$ (when $\mathcal{S}(x)$ is already built).

**COROLLARY 7.** Given two words x and t on a finite alphabet A, LCF(x,t) can be computed in time and space complexities $O(|xt|)$.

The use of suffix transducers instead of suffix automata allows to memorize an occurrence of a longest common factor, the first for instance.

```
begin {states (q and i) and transitions are those of $\mathcal{S}(x)$.
       On states are defined functions s ans l}
   k←0 ; $l_0$←0 ; q←i ;
   while not end of input t do
      read next letter a ; k←k+1 ;
      if q.a defined then
         $l_k$←$l_{k-1}$+1 ; q←q.a ;
      else
         while q≠i and q.a not defined do
            q←s(q) ;
         end while ;
         if q.a not defined then $l_k$←0 ;
         else
            $l_k$←l(q)+1 ; q←q.a ;
         end if ;
      end if ;
   end while ;
end.
```

**FIGURE 3.** **Computing lengthes of factors common to x and t.**

## REAL-TIME STRING-MATCHING.

The algorithm of the preceeding section has a linear time complexity but the delay between the reading of two consecutive letters of the input text t depends on the word x and is even $O(|x|)$ in the worst case. By considering a new suffix link, the delay can be bounded to $O(|A|)$.

First define the immediate right context of a factor u of word x to be the set of letters that follow u :

$$C(u)=\{a \in A / \exists y,z \in A^* \ x=yuaz\}.$$

When v is a suffix of u we get $C(u) \subseteq C(v)$ and in particular $C(u) \subseteq C(s(u))$ when $u \neq 1$.

When $C(u)=C(s(u))$ and if ub is encountered in the text t which is searched for x, then it is useless to come down to s(u)b. This gives the idea of a new suffix link noted sa and defined on non empty factors u of x :

$$sa(u) = \begin{cases} s^k(u) \text{ if } k \text{ is the smallest integer} > 0 \\ \qquad \text{such that } C(s^k(q)) \neq C(u) \\ 1 \quad \text{otherwise} \end{cases}$$

Since $C(w) \subseteq C(s(w))$, the test $C(s^k(u)) \neq C(u)$ can be done on the cardinalities of the two sets. The condition on immediate right context easily translates on automaton $\mathcal{S}(x)$ in term of output degree de of its states. If q is a state of $\mathcal{S}(w)$, let

$$de(q)=|\{a \in A / q.a \text{ defined}\}|.$$

Then the new suffix link sa on states of $\mathcal{S}(w)$ (different from initial state i) is

$$sa(q)= \begin{cases} s^k(q) \text{ if } k \text{ is the smallest integer} > 0 \\ \qquad \text{such that } de(s^k(q)) \neq de(q), \\ i \quad \text{otherwise.} \end{cases}$$

**PROPOSITION 8.** Replacing instruction '$q \leftarrow s(q)$' by '$q \leftarrow sa(q)$' in the algorithm of figure 3 leads to a real-time algorithm on any finite alphabet.

To compute the suffix link sa, states of $\mathcal{S}(x)$ are visited in a breadth-first-search order and the following formula is applied which yields an $O(|x|)$ time complexity algorithm.

$$sa(q)= \begin{cases} i & \text{if } s(q)=i, \\ s(q) & \text{if } de(s(q)) \neq de(q), \\ sa(s(q)) & \text{otherwise.} \end{cases}$$

Another way to get a real-time string-matching is to complete the transducer $\mathcal{S}(x)$. In fact the aim of the internal 'while' loop of algorithm in figure 3 and the test hereafter is to compute missing transitions. With complete transducers the space complexity becomes $O(|A|.|x|)$ while it is $O(|x|)$ in the previous algorithm.

**MAIN REFERENCES** :

[Be 79] J. BERSTEL,Transductions and context-free languages, Teubner, 1979.

[BBEHCS 85] A. BLUMER, J. EHRENFEUCHT, D. HAUSSLER, M.T. CHEN & J. SEIFEIRAS,
The smallest automaton recognizing the subwords of a text,
Theor. Comput. Sci. 40, 1 (1985) 31-56.

[Ch 83] C. CHOFFRUT,
On some combinatorial properties of metrics over the free monoid,
in : [Combinatorics on words, Cumming ed., Academic Press, 1983].

[Cr 86] M. CROCHEMORE,
Transducers and repetitions,
Theor. Comput. Sci. (1986) to appear.

[KMP 77] D.E. KNUTH, J.H. MORRIS & V.R. PRATT,
Fast pattern-matching in strings,
SIAM J. Compt. 6, 2 (1977) 323-350.

[Sl 83] A.O. SLISENKO,
Detection of periodicities and string-matching in real time,
J. of Soviet Mathematics 22, 3 (1983) 1316-1387.