1985

Structural Properties of the String Statistics Problem

A. Apostolico

F. P. Preparata

Report Number:
85-541

STRUCTURAL PROPERTIES OF THE STRING STATISTICS PROBLEM

A. Apostolico

F. P. Preparata
University of Illinois

CSD-TR-541
October 1985

STRUCTURAL PROPERTIES OF THE STRING STATISTICS PROBLEM*

A. APOSTOLICO
Department of Computer Sciences
Purdue University
West Lafayette, IN 47907

F.P. PREPARATA
Coordinated Sciences Laboratory
University of Illinois at Urbana-Champaign
Urbana, Illinois 61801

* This work was supported in part by the Italian Ministry of Education Scientific Program. Additional support was provided by NATO under research grant 039.82.

Running Head: STRUCTURE OF THE STRING STATISTICS PROBLEM

Mailing Address for Proofs:
Prof. Alberto Apostolico
Dept. of Computer Science
Purdue University
West Lafayette, In. 47907

ABSTRACT

A suitably weighted Index Tree such as a B-tree or a Suffix Tree can be easily adapted to store, for a given string x and for all substrings w of x, the number of distinct instances of w along x. The storage needed is seen to be linear in the length of x: moreover, the whole statistics can itself be derived in linear time, off-line of a RAM.

If the substring w has nontrivial periods, however, the number of distinct instances might differ from that of distinct *nonoverlapping* occurrences along x. It is shown here that $O(n \log n)$ storage units - n standing for the length of x - are sufficient to organize this second kind of statistics, in such a way that the maximum number of nonoverlapping instances for arbitrary w along x can be retrieved in a number of character comparisons not exceeding the length of w.

SYMBOLS

This paper was typeset in UNIX-TROFF using standard Times Roman font for the text. Most symbols are Italics with Italics subscript and/or superscripts. Font style examples are reported in the table of next page.

Attention is called on the following special symbols:

$\alpha, \beta, \gamma, \vartheta, \mu, \nu$ are Greek, lower case alpha, beta, gamma, theta, mu, nu.

$\mathbb{N}$ is a special upper case 'N' used to denote the cardinality of the set of the integers.

$\stackrel{\triangle}{=}$ is 'mathematical equal' with a superimposed triangle; this is used to denote 'equal by definition'.

□ is a small square used to mark the end of proofs.

$\in$ is the 'member of' sign and should not be confused with ε.

The $\lfloor$ and $\rfloor$ ('left floor' and 'right floor', i.e., left and right bottom of big square bracket) signs should not be confused with square brackets.

EXAMPLE FONTS

Times Roman

abcdefghijklmnopqrstuvwxyz
ABCDEFGHIJKLMNOPQRSTUVWXYZ
1234567890
! $ % & ( ) ' ' * + − . , / : ; = ? [ ] |
• □ — - _ ¼ ½ ¾ fi fl ff ffi ffl ° † ' ¢ ® ©

Times Italic

abcdefghijklmnopqrstuvwxyz
ABCDEFGHIJKLMNOPQRSTUVWXYZ
1234567890
*! $ % & ( ) ' ' * + − . , / : ; = ? [ ] |*
• □ — - _ ¼ ½ ¾ fi fl ff ffi ffl ° † ' ¢ ® ©

Times Bold

abcdefghijklmnopqrstuvwxyz
ABCDEFGHIJKLMNOPQRSTUVWXYZ
1234567890
! $ % & ( ) ' ' * + − . , / : ; = ? [ ] |
• □ — - _ ¼ ½ ¾ fi fl ff ffi ffl ° † ' ¢ ® ©

Special Mathematical Font

" ' \ ^ _ ` ~ / < > { } # @ ÷ − = ▪
α β γ δ ϵ ζ η θ ι κ λ μ ν ξ ο π ρ σ ς τ υ φ χ ψ ω
Γ Δ Θ Λ Ξ Π Σ Υ Φ Ψ Ω
√ ⩾ ⩽ ≡ ~ ≃ ≠ — — ∣ ∥ × ÷ ± ∪ ∩ ⊂ ⊃ ⊆ ⊇ ∞ ∂
§ ▽ ¬ ∫ ∝ ∅ ∈ ‡ ☛ ☚ Ⓐ ⌇ ○ ⟮ ⟯ ⟭ ⦃ ⦄ ⟦ ⟧ ⟦ ⟧

FIGURE CAPTIONS

Figure 1: The suffix tree of the string *abbaabb$*

Figure 2: Weighted Trees for $(st)^5s$ (partial).
(a) Suffix-tree-with-weights-from-C_1
(b) Suffix tree (noncompact version) with weights from C_2
(c) Minimal Augmented Suffix Tree with weights from C_2.

Figure 3: No two auxiliary nodes from N_2 can fall on the same original arc of T_x.

Figure 4: Illustration for Example 2.

Figure 5: The extraction of necklaces from a run of segments.

Figure 6: The two necklaces extracted from the run of Fig. 5.

Figure 7: A run may contain more than one chunk.

Figure 8: Assuming that two consecutive chunks at α overlap on more than $p-1$ positions generates a contradiction.

Figure 9: The effect of the contraction of segments in a necklace.

STRUCTURAL PROPERTIES OF THE STRING STATISTICS PROBLEM*

A. APOSTOLICO
Department of Computer Sciences
Purdue University
West Lafayette, IN 47907

F.P. PREPARATA
Coordinated Sciences Laboratory
University of Illinois at Urbana-Champaign
Urbana, Illinois 61801

September, 1983
Revised, December 1984

ABSTRACT

A suitably weighted Index Tree such as a B-tree or a Suffix Tree can be easily adapted to store, for a given string x and for all substrings w of x, the number of distinct instances of w along x. The storage needed is seen to be linear in the length of x: moreover, the whole statistics can itself be derived in linear time, off-line of a RAM.

If the substring w has nontrivial periods, however, the number of distinct instances might differ from that of distinct *nonoverlapping* occurrences along x. It is shown here that $O(n \log n)$ storage units - n standing for the length of x - are sufficient to organize this second kind of statistics, in such a way that the maximum number of nonoverlapping instances for arbitrary w along x can be retrieved in a number of character comparisons not exceeding the length of w.

Key words and Phrases: statistics in a textstring, pattern matching, suffix trees, augmented suffix trees, period of a string, repetitions in a string.

* This work was supported in part by the Italian Ministry of Education Scientific Program. Additional support was provided by NATO under research grant 039.82.

1. INTRODUCTION

Clever pattern matching techniques and tools [AH,KM,BM,GA,AG] have been developed in recent years to count (and locate) all distinct occurrences of an assigned substring w (the *pattern*) within a string x (the *text*). As is well known, this problem can be solved in $O(|x|)$ time, regardless of whether instances of the same pattern w that overlap - i.e., share positions in x - have to be distinctly detected, or else the search is limited to one of the streams of consecutive nonoverlapping occurrences of w.

When frequent queries of this kind are in order on a fixed text, each query involving a different pattern, it might be convenient to preprocess x to construct an auxiliary index tree [AH,WE,MC,MR] storing in $O(|x|)$ space information about the structure of x. This auxiliary tree is to be exploited during the searches as the state transition diagram of a finite automation, whose input is the pattern being sought, and requires only time linear in the length of the pattern to know whether or not the latter is a substring of x. It will become apparent in the following that some simple additional manipulations on the tree make it possible to count the number of distinct (possibly overlapping) instances of any pattern w in x in $O(|w|)$ steps. In other words, the full statistics (with possible overlaps) of the substrings of a given string x can be precomputed in one of these trees, within time and space linear in the textlength.

Contrary to the single-substring case, the efficient computation and storage of the full statistics without overlaps is a more difficult problem. Apart from its purely combinatorial interest, this problem is relevant in a variety of computer applications in computational linguistics, data compression, text editing, pattern recognition, signal processing, etc.

In Sections 2-4 of this paper, we present a structure, derived from suffix

trees [MC], which collects in $O(|x|\log|x|)$ storage units all distinct substrings of x in such a way as to make it possible to know, for each such substring w and in $|w|$ comparisons, the maximum number of templates of w that can be aligned on x so as to match the textstring while not overlapping with one another. It is worth to point out that the actual alignment of the templates along the textstring might not be unique.

We show that the proposed index, that we call Augmented Suffix Tree (AST, for short), is unique in its minimal form for any assigned string. Thus the rest of the paper is devoted to the analysis of the structure of the minimal AST associated with an arbitrary string. As indicated earlier, the AST is constructively viewed as a modification of the standard suffix tree, carried out in a bottom-up process that constructs the AST by merging two ASTs and by inserting the extra nodes required by the change in statistics. Although this process is readily carried out in time $O(|x|^2)$, in a subsequent paper we shall present a algorithmic technique achieving the same objective in time $o(|x|^2)$.

2. PRELIMINARIES

Let I be a finite alphabet and I^+ the free semigroup generated by I. A *string* $x \in I^+$ is fully specified by writing $w = a_0 a_1 \cdots a_{n-1}$, where $a_i \in I$ $(i = 0,1,\ldots,n-1)$ and $|x|$ denotes the length of x.* We assume here that x is stored as an array $x[0{:}n-1]$, where $x[i] = a_i (i = 0,1,\ldots,n-1)$. Given $x = a_0 a_1 \cdots a_{n-1}$, w is a *substring* of x if there exist indices i,j $(0 \le i \le j \le n-1)$ such that $w = a_i a_{i+1} \cdots a_j$. A *factor* of x is a substring of x and its starting index in $(0,1,\ldots,n-1)$ (that is, a positioned substring). The notation $x(i,j)$ is used to denote the factor of $x{:}x[i]x[i+1]\cdots x[j]$. A left (right) factor of x is a *prefix (suffix)* of x. Two factors $x(i,j)$ and $x(m,h)$ are *equivalent* if their

* Occasionally in what follows it will be assumed implicitly that $|x| = n$.

associated substrings are identical. Moreover, two equivalent factors $x(i,j)$ and $x(m,h)$ are said to *overlap* if either

$$m \le j < h \quad \text{or} \quad i \le h < j .$$

The set of all distinct nonempty substrings of x (*words*) is called the *vocabulary* of x and denoted by V_x. A *weighted vocabulary* for x is any pair (V_x, C), where $C: V_x \to \mathbb{N}$ is a mapping that associates with each string $w \in V_x$ a natural number $k = C(w)$.

This work is devoted to the study of two particular weighted vocabularies. Namely, C_1 associates with each $w \in V_x$ the number of distinct equivalent factors of x that correspond to w; on the other hand, C_2 associates with each $w \in V_x$ the maximum number of distinct factors $x(i_1,j_1), x(i_2,j_2), \ldots, x(i_p,j_p)$ corresponding to w and such that it is possible to write $x = w_1 w w_2 w w_3 \cdots w w_{p+1}$ with $w_d \in I^*$ $(d = 1,2,\ldots,p+1)$.

In the following, we will refer to the pairs (V_x, C_1) and (V_x, C_2) as to the *statistics of type 1 (or with overlaps)* and *type 2 (or without overlaps)*, respectively.

3. STRUCTURING WEIGHTED VOCABULARIES

As is well known, there can be as many as $O(n^2)$ distinct words in a string x of length n. This is certainly the case if no two distinct factors of x are equivalent. In practice, however, the number of distinct substrings is often less than the number of factors, since imposing that the latter be all different implies that the string itself reduces to one of the permutations of the symbols in some subset $\bar{I}$ of I. In the other extreme, the string $x = a^n$ has $|V_x| = n$: in this case the string itself is a suitable representation for V_x, and $O(n)$ storage suffices for any pair (V_x, C) as induced by an arbitrarily chosen set of weights.

Neglecting for a moment the weights in C, consider first the problem of organizing in a compact way the distinct words of V_x. Letting $\$$ be a special symbol not included in I, V_x can be conveniently stored into the so-called *suffix tree* [AA, AL, MC] T_x for $x\$$. As is well known, such a tree T_x is rooted, has $O(n)$ nodes and for a string $x\$$ is defined as follows: Each arc is associated with a word in V_x by names of a suitable factor of $x(O,n)$, and each path from the root to a leaf describes the suffix obtained by concatenating the substrings associated with the sequence of its arcs. Thus, if $x\$$ is stored in $x[O:n]$, a leaf of T_x is labelled with the integer j if the corresponding path describes the suffix $x(j,n)$. An arc is labelled by an ordered pair (i,j) $(i \leq j)$ if the associated substring is identical to the substring of the factor $x(i,j)$.

Although a brute force approach would use $O(n^2)$ operations to construct T_x for $|x| = n$, there exist clever algorithms for its construction in linear time [AH,WE,MC].

Any vertex α of T_x distinct from the root describes a substring $W(\alpha)$ of x in a natural way (the concatenation of the factors associated with the arcs leading to α from the root); vertex α is called the *proper locus* of $W(\alpha)$. In general, for any $w \in V_x$, the *locus* α of w is the unique vertex of T_x such that w is a prefix of $W(\alpha)$ and $W(\text{FATHER}(\alpha))$ is a proper prefix of w. It follows from the definition of T_x that for any substring w of x whose locus is α, the number of distinct occurrences of w in x (the number of equivalent factors associated with w) is equal to the number of leaves of the subtree of T_x rooted at α. In addition, the labels of the leaves of this subtree completely identify the positions of the first symbols of all factors whose substrings are identical to w.

Once T_x is used to store V_x, the set of weights C_1 can be readily computed and stored into the tree itself in a straightforward way: indeed, it will suffice to visit the tree in post-order while evaluating the locally defined function that

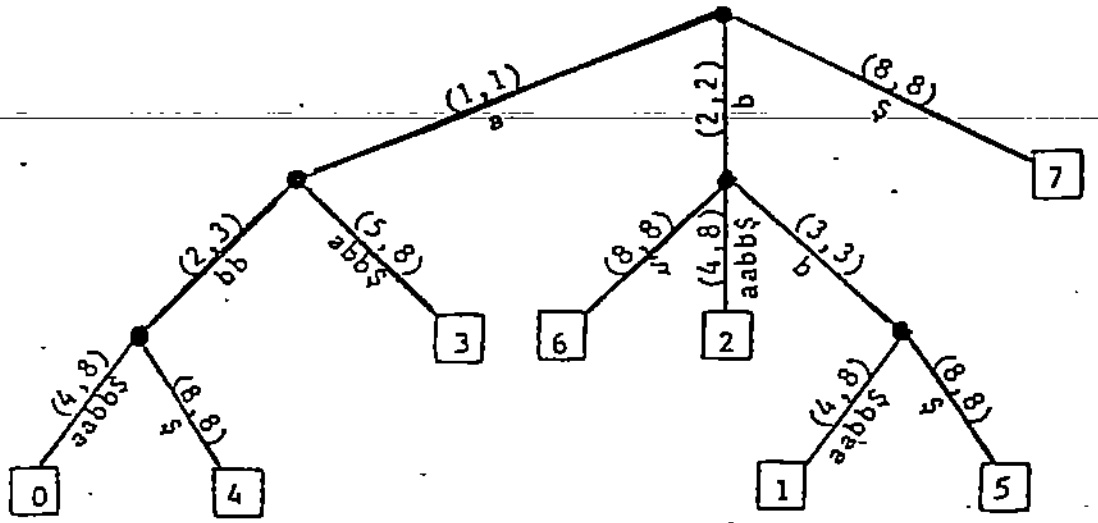

Figure 1. The suffix tree of the string *abbaabb$*

associates with each node in T_x the number of leaves of the subtree rooted at that node. If each node α is provided a special field to store the value $C_1(w)$ pertaining to the word w of x such that $W(\alpha) = w$, the resulting weighed tree can be exploited to know, for any arbitrary pattern $v \in I^+$ and in a number of comparisons proportional to $|v|$, the number of instances of v that are found in x.*

Indeed, starting from the root of T_x, we scan the downward path in response to the symbols of v. If, at any point in the process, a mismatch is detected, then only a prefix of v appears in x. Otherwise, after the last symbol of v has been matched, then the locus α of v contains the weight $C_1(v)$. Notice that this is true regardless of whether α is the proper locus of v.

The organization of (V_x, C_2) along the same lines is not so easy. In fact, it is not obvious that the original $O(n)$ nodes in T_x shall suffice in general to carry all

* To avoid unnecessary burden in notations we will make no distinction, here and hereafter, between original trees and their weighted versions.

needed information. To make this point clearer, consider the following example (refer to Figure 2). Let $x = (st)^5s$, and assume for simplicity that $s,t \in I$. Consider now the portion of T_x that corresponds to all substrings in the form $(st)^k s$ ($k = 0,1,\ldots,5$). The vertices of T_x in Figure 2(a) have been labelled with the weights that the previously described computation would ordinarily attribute to them. Such labels are not needed of course on the leaves, which are loci of substrings that occur only once within x, by construction. Figure 2(b) displays the same part of the tree T_x with additional internal nodes, which are used to store the values of C_2 that could not be accommodated in the original tree. The additional nodes have degree one and might force the overall number of vertices in the tree to be $O(|x|^2)$ in the worst case [AH]. However, an inspection of the figure also shows that not all of the extra nodes are strictly necessary: in fact, if we retain only those extra nodes whose C_2-value is different from that of the subsequent original node, then just the locus of st is kept, as shown in Figure 2(c). Notice that the same argument still holds if s and t are assumed to be arbitrary words in I^*, provided that st is not of the form $(v)^p$, for some $v \in I^+$ and $p \geq 2$.

We define an *Augmented Suffix Tree* (AST) for the string x, denoted $\overline{T}_x$, as any suffix tree for x that has been expanded with extra nodes in such a way that, for any word $u \in V_x$, the locus α of u is labelled with $C_2(u)$. Notice the similarity between an AST, such as the one depicted above for string $(st)^5s$, and the position tree [AH] for the string in its non-compact version [AH,WE].

Further, we say that an AST is *minimal* if the removal of any of its internal nodes causes the resulting structure not to be an AST any longer. It is easy to show that, if T_x is minimal, then the C_2 value of any of the vertices of degree one must differ from the one associated with its (unique) son.

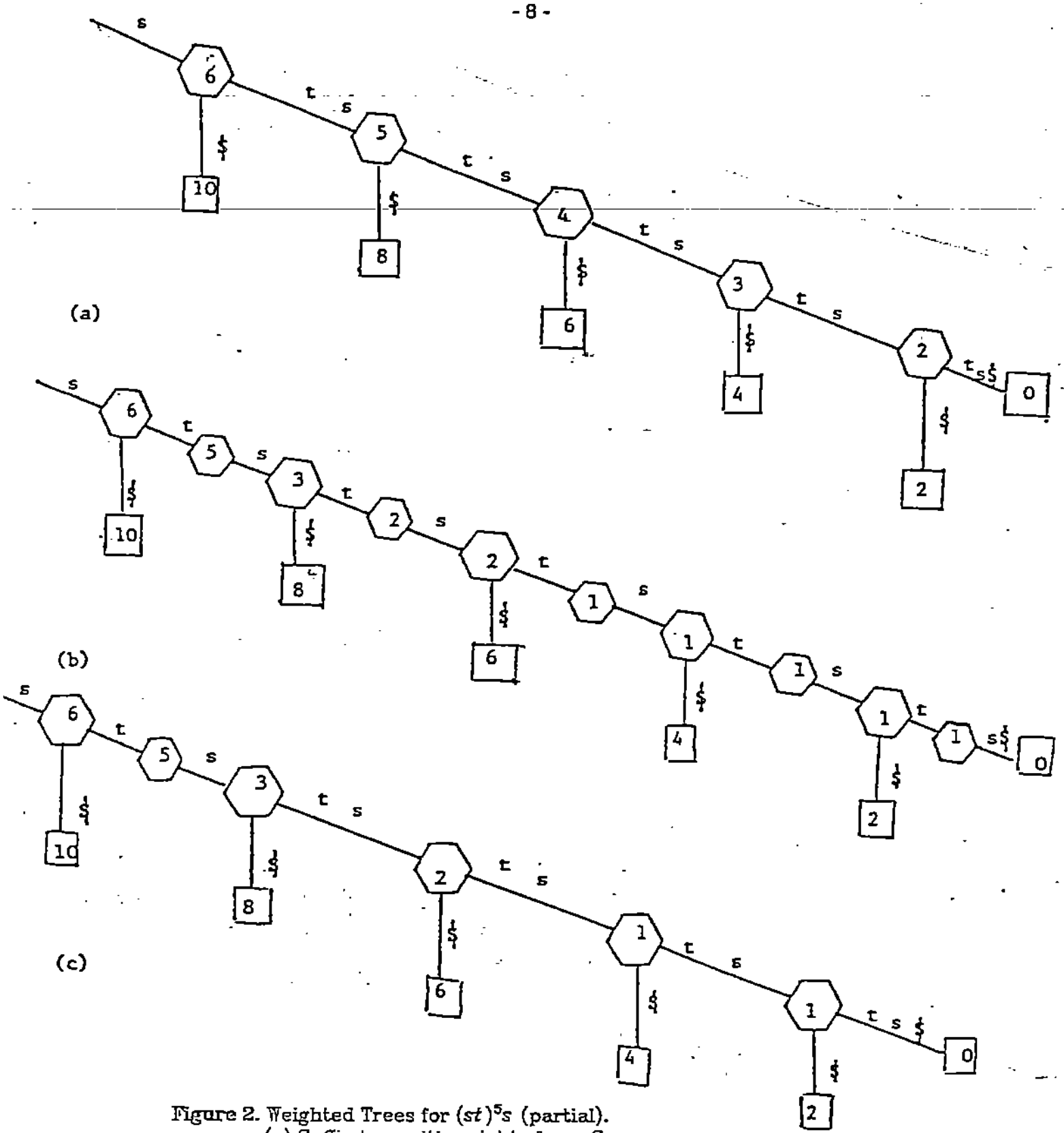

Figure 2. Weighted Trees for $(st)^5s$ (partial).
(a) Suffix tree with weights from C_1
(b) Suffix tree (non compact version) with weights from C_2
(c) Minimal Augmented Suffix Tree with weights from C_2

Theorem 1: For any $x \in I^+$, there is a unique minimal AST for x.

Proof:

Assume by contradiction that $\hat{T}_x^{(1)}$ and $\hat{T}_x^{(2)}$ are both minimal AST's for x. Since the underlying suffix trees must be identical, then they must differ in the nodes of degree 1. Let α_1 be one such node in, say, $\hat{T}_x^{(1)}$, with no homologous node in $\hat{T}_x^{(2)}$ and let $v = W(\alpha_1)$ be the string in V_x whose locus is α_1. By the minimality of $\hat{T}_x^{(1)}$ if a is the symbol following $W(\alpha_1)$, then the locus of the string va must be labelled with a weight different from the one associated with α_1. On the other hand, v and va share the same locus (and weights) in $\hat{T}_x^{(2)}$, so that the two trees are not both correct AST's for the string x. □

Hereafter, we will use the notation $\hat{T}_x$ to indicate the unique minimal AST associated with the string x.

4. MINIMAL AUGMENTED SUFFIX TREES

In this section, we will show that $O(n \log n)$ storage suffices to store the minimal AST $\hat{T}_x$ associated with any given string x of length n.

In order to derive this property of $\hat{T}_x$, however, some definitions and background results are to be reviewed.

An integer p is a *period* of x if $x(i) = x(i+p)$ $(i = 1,2,\ldots |x|-p)$. A string x is *periodic* if it has a period of size not larger than $|x|/2$. Let x be periodic and let p be its smallest period: the prefix (suffix) of x length p is called the *left (right) root* of x with respect to p. We remark that the notions of periodicity and overlap among equivalent factors are closely related: indeed it is easily seen

[KM, LO]) that the two factors $x(1,d)$ and $x(j,n)$, $j \leq d+1$, are equivalent iff there are $k \geq 2$, $s \in I^*$ and $t \in I^+$, such that $x = (st)^k s$, with $|st| = j-1$.

Recall also that a string $x \in I^+$ is *primitive* if setting $x = u^k$ implies $u = x$ and $k = 1$. It is a simple exercise to show that, with the aid of a suffix tree, we can decide in linear time if a string is primitive (or if any of its prefixes is not). A string $x \in I^+$ is *strongly primitive* or *square-free* if, expressing x as $x = v_1 u^k v_2$, with $u \in I^+$ and $v_1, v_2 \in I^*$, implies $k = 1$. Equivalently, x is square-free if and only if each $w \in V_x$ is primitive. Notice that the minimal AST for a square-free x reduces to the suffix tree T_x so that the type 1 and type 2 statistics for x also coincide. However, to decide efficiently whether a string is square-free is a rather complicated problem. Indeed, although $O(n^2)$-time algorithms can be readily developed on the basis of existing pattern matching tools [AH,KM,BM,GA,AG], optimal algorithms for the cases where the alphabet size can be regarded as a constant have been introduced only recently [CM, LM] (cfr. also [AL]).

A *repetition* in x is a factor $x(i,m)$ for which there are indices j,d $(i < d \leq j \leq m)$ such that: (a) $x(i,j)$ is equivalent to $x(d,m)$; (b) $x(i,d-1)$ corresponds to a primitive word; (c) $x[j+1] \neq x[m+1]$. It is easily seen that the notion of repetition is also closely related to overlaps among equivalent factors. Indeed, a repetition is a periodic factor in the form $(st)^k s$ where $k > 1$, $s \in I^*$ and $t \in I^+$; as such, it is completely identified by the triple of its starting position i, its period $p = d-1$, and its length $L = m - i + 1$, respectively, and is denoted by the symbol $R(i,p,L)$.

Efficient search for all distinct repetitions in a string is somewhat more involved than simply testing for string square-freedom: three strategies have been developed, to date, which detect all distinct repetitions in a string in time $O(n \log n)$ [CR,AP,ML]. As is well-known, the number of distinct repetitions in

some string is lower-bounded by $n \log n$ [CR], so that, trivially, the running times of the above algorithms are also asymptotically optimal.

We turn now to the minimal augmented suffix tree (MAST) $\hat{T}_x$ for x. As the above discussion suggests, repetitions are responsible for the additional nodes inserted in T_x to obtain $\hat{T}_x$. In $\hat{T}_x$, we call *original* the nodes also present in T_x and *auxiliary* all the additional nodes.

Theorem 2: If α is an auxiliary node of $\hat{T}_x$, then there are substrings u, $v \in V_x$ and an integer $k \geq 1$ such that $W(\alpha) = u = v^k$ and there is a repetition in x in the form $v^m v'$ with v' a prefix of v and $m \geq 2k$.

Proof:

Indeed, letting in $\hat{T}_x$ μ and ν be the father and son node of α, respectively, by the definition we have $C_2(W(\mu)) \geq C_2(W(\alpha)) \geq C_2(W(\nu))$. Letting now $w = W(\nu)$, there must be a symbol $a \in I$ such that $w = uar$, with $r \in I^*$. Since α has outdegree one, then all instances of u (ua) in x, whether overlapping with one another or not, occur as prefixes of w. Therefore, $C_2(u)$ can differ from $C_2(w)$ only if some pair(s) of consecutive overlapping instances of w host two overlapping occurrences of ua whose prefixes u do not overlap anymore. Hence $u^2 \in V_x$; obviously u^2 must correspond in x to the prefix of a repetition which has the form $u^2 u^p u'$ or $v^{2k} v^{kp} u'$ depending on whether u is primitive or not. □

Based on the n log n bound on the number of distinct repetitions in x, it is not difficult to derive that the minimal AST associated with x has a number of nodes bounded by $O(n \log n)$. We choose to give here a proof of this bound that sheds additional information on the distribution of auxiliary nodes in $\hat{T}_x$.

Auxiliary nodes are further subdivided into two classes, N_1 and N_2, according to the following property: a node α belongs to N_1 if it is the locus of the root u of some repetition in x, otherwise it belongs to N_2. Notice by Theorem 2, that if $\alpha \in N_2$, then, by Theorem 2, it is the locus of a substring in the form u^k, for some primitive u and $k > 1$.

Theorem 3: The minimal AST $\hat{T}_x$ for x has $O(n \log n)$ auxiliary nodes.

Proof:

To prove the claim, we make use of the following well-known *periodicity* Lemma [LS,LE]:

Lemma:

If $w \in I^+$ has periods p and q, and $|w| \geq p+1$, then w has period $gcd(p,q)$.

To start with the proof, let $R(i,p,L)$ be one of the repetitions in x and let u be its (primitive) left root. If the repetition R is in the form u^k for some $k > 1$, then it is easily seen that no auxiliary nodes are needed in $\hat{T}_x$ on the path from the root to the locus of $x(i,n)$. Therefore, it will be assumed that R be in the form $(st)^k s$ with $k > 1$ and $t,s \in I^+$.

Consider the nodes in N_1 first. The number of distinct factors that correspond to substrings of the form u^2 equals the number of distinct repetitions in x, and this last number is bounded by $n \log n$. Therefore, the number of distinct roots of repetitions in x cannot exceed this bound. Whence the cardinality of N_1 is at most $n \log n$.

Next, consider the nodes in N_2. By definition, if β is such a node then there must be a factor in x that corresponds to a word in the form $W(\beta)W(\beta)$. We claim that there cannot be two nodes of N_2 in $\hat{T}_x$ that are inserted into the same arc of T_x.

In fact, assume for a contradiction, that there are two such nodes, μ and ν, inserted into the same original arc from ρ to τ in T_x (see Figure 3). Since μ and ν are auxiliary nodes in N_2, then it must be:

$$W(\mu) = u^k$$

$$W(\nu) = v^d$$

for some $u, v \in I^+$, u, v primitive and $k, d \geq 2$.

But we also notice that by construction, $W(\nu) = W(\mu)\tau$ for some $\tau \in I^+$, that is, $v^d = u^k\tau$. Now τ must be a prefix of u: indeed since μ is an auxiliary node in $\hat{T}_x$, its associated weight in C_2 is such that $C_2(\mu) \geq 2$, which means that there is at last one occurrence of u^{2k} in x, whose locus must fall on the leafwards extension of the path from the root of ν. Therefore, $|v^d| = d|v| > |v| + |u|$, so that $W(\nu)$ has periods $|u|$ and $|v|$ at the same time, which contradicts the hypothesis that v be primitive.

In conclusion, any two auxiliary nodes in N_2 in $\hat{T}_x$ appear in distinct original arcs of T_x, whence the cardinality of N_2 is bounded by $O(n)$. This completes the proof. □

Having established individual bounds for the sizes of N_1 and N_2 is useful in studying MAST's of special classes of strings. For instance, consider the class of strings x such that $u^2 \in V_x$, with u primitive, implies also that $u^3 \in V_x$. By using the periodicity Lemma as in Theorem 3 above, it can be shown [AA] that the size of set N_1 is bounded by n for such strings, whence their MAST's need only $O(n)$ storage.

We conclude this section by remarking that it is an open problem to determine whether there are MASTs that actually require $\vartheta(n \log n)$ space.

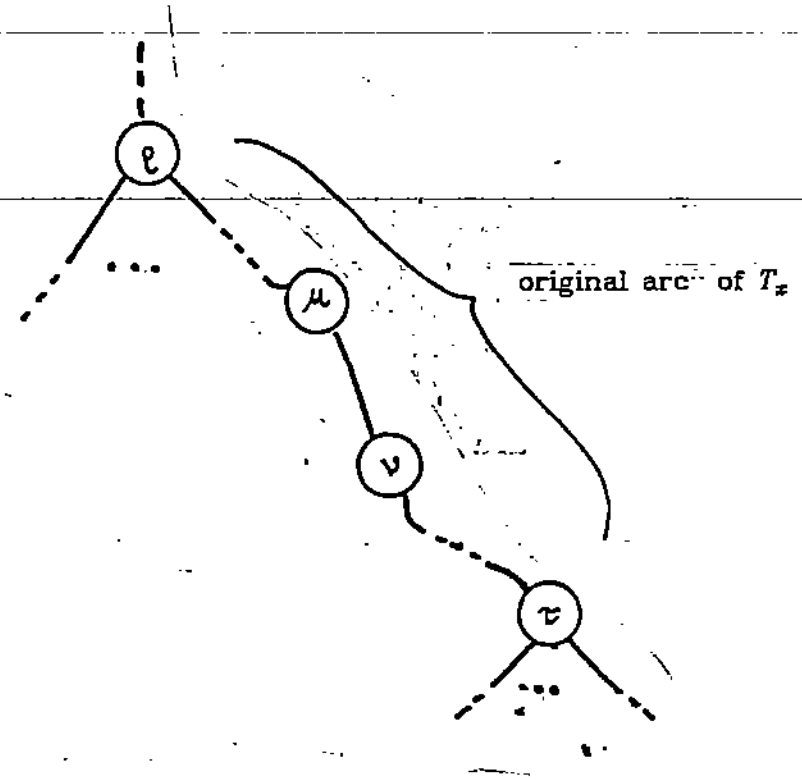

Figure 3.

Figure 3.

No two auxiliary nodes from N_2 can fall

on the same original arc of T_x.

5. CONSTRUCTING MINIMAL ASTs

The preceding discussion on the structure of MASTs naturally suggests the following construction algorithm:

A. Construction of the Suffix Tree T_x for input string x.

B. Detection of all repetitions in x with consequent introduction, in T_x, of an auxiliary node α any time there happens to be a factor of x in the form v^2 and v has no proper locus in the tree.

C. Weighting of nodes in the resulting augmented tree, and deletion of each auxiliary node whose weight is found to be identical to that of its (unique) son.

As mentioned earlier, step A can be carried in $O(n)$ time by known methods [AH,WE,MC]. It was shown in [AP] that the suffix tree T_x itself can be used to detect in optimal $O(n \log n)$ time and space $O(n)$ - all distinct repetitions in x. This is done by a rather sophisticated bottom-up merge of leaves in T_x and by exploiting the following simple property of T_x:

Lemma 0 [AP]: $R(i,p,L)$ is a repetition of x if and only if there is a vertex α in T_x such that $|W(\alpha)| \geq p$, where i and $j = i+p$ are consecutive leaves in the subtree of T_x rooted at α.

We shall see in Section 7 how this process can be modified so as to insert additional nodes on-the-fly from a son to a father node, whenever needed. However, the discussion of step C requires some preliminary considerations on the actual computation of C_2-values associated with internal nodes of T_x.

Let v be an arbitrary pattern of x. We call *v-tagging* of x any maximal set of nonoverlapping factors of x all of which correspond to v; in addition, we call *compact v-tagging* of x, and denote it by P, the unique maximal set $x(i_1,j_1)$; $x(i_2,j_2)$;...$x(i_k,j_k)$, of equivalent factors such that $x(i_{d+1},j_{d+1})$ is the instance of v closest to $x(i_d,j_d)$ and not overlapping with it ($d = 1,2,\ldots,k$).

Example 1: Let $x = b\ a\ b\ a\ b\ a\ b\ a\ b\ a\ b\ b\ a\ b\ a\ b\ a$, and $v = a\ b\ a$. Then

```
b a b a b a b a b a b b a b a b a
  a b a       a b a         a b a
```

and

```
b a b a b a b a b a b b a b a b a
      a b a   a b a     a b a
```

are both v-taggings of x. The unique compact v-tagging of x is:

```
b a b a b a b a b b a b a b a
  a b a   a b a         a b a
```

Theorem 4: Let k be the cardinality of the compact v-tagging of x. Then $C_2(v) = k$.

Proof:

Indeed, it is obviously $C_2(v) \geq k$. Let then P' be any *v-tagging* with $|P'| = m$, and consider the ordered sets $Q \equiv \{P - P \cap P'\}$ and $Q' \equiv \{P' - P \cap P'\}$. By construction, however we choose an element $x(i_r', j_r')$ from Q', there must be a corresponding element $x(i_r, j_r)$ in Q such that $j_r < i_r' < i_r$. Moreover, consecutive elements of Q (Q') do not overlap, whence $|Q'| \leq |Q|$ and $m \leq k$. □

The above discussion makes it natural to organize the C_2 weighting process of the tree as a new bottom-up computation on sorted lists of leaves: the merging at node α of the sorted lists of its offsprings can also be used to construct the compact *$W(\alpha)$-tagging* of x. In addition, steps (B) and (C) above could be easily combined into a unique bottom-up computation.

In any case, the weighting of internal nodes appears to be the most time-consuming operation, as it may require $\vartheta(n^2)$ steps in the worst case. In order to improve over such a performance, we have to study more in depth the structure and evolution of compact taggings.

6. RUNS, CHUNKS AND NECKLACES

Let $S(\alpha)$ be the ordered sequence of leaves in the subtree of T_x rooted at vertex α, and let i and j be two elements of $S(\alpha)$, with $i < j$. *Segment i* is the factor $W(\alpha)$ starting at i. Segments i and j are said to *overlap* if $j - i < |W(\alpha)|$. Moreover, if segments i and j overlap and are consecutive, then leaf i is called the *origin* for j and leaf j is called the *detector* for i. Notice that two

overlapping leaves need not be consecutive in $S(\alpha)$, and that a leaf can be an origin and a detector at the same time, as shown in the following example:

Example 2: Consider the string of Fig. 4.

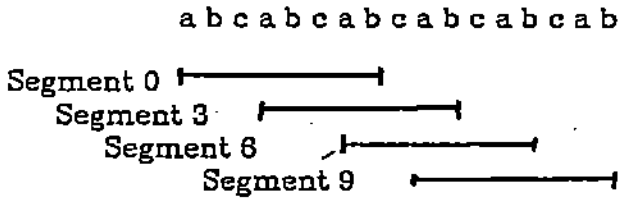

Figure 4 –
Illustration for Example 2

With reference to that figure, let $W(\alpha) = abc\ \ abc\ \ ab$. Then leaves 0,3,6, and 9 are in the subtree of T_α rooted at α. Clearly, segment 6 and segment 0 overlap, although they are not consecutive. In addition, leaf 3 is the origin for 6 and the detector for 0.

It will be necessary to consider sequences of overlapping segments. To this end, we introduce the notion of "runs" of segments (at α) as follows:

Definition 1: If $i_0, i_1 \cdots i_{s+1}$ is a substring of $S(\alpha)$ and $i_1 - i_0 \geq |W(\alpha)|$, $i_{j+1} - i_j < |W(\alpha)|$ for $j = 1,2, \ldots, s-1$, and $i_{s+1} - i_s \geq |W(\alpha)|$ the sequence of segments $i_1, i_2, \ldots, i_s$ is a *run*.

Segments i_1 and i_s are called, respectively, the *head* and the *tail* of the run.

The *span* of the run is the interval which is the union of the segments in the run.

Within a run we single out the following important subsequence of segments:

Definition 2: A *necklace* is a maximal subsequence of segments in a run such that only consecutive segments overlap.

Necklaces are extracted from a run by means of a simple scan of it. The input run is described as a doubly-connected list with pointers SUCC and PRED and terminal item * (in addition, the dummy predecessor of the first segment is supposed to be nonoverlapping with it). Necklaces η_1, η_2 are queues.

```
1.  begin s ← first segment of input run;
2.      j ← 1;
3.      while s ≠ * do (/ the scan continues /)
4.          begin η_j ← s;
5.              if SUCC[s] ≠ * do
6.                  while SUCC[s] overlaps do
7.                      if SUCC[s] overlaps PRED[s] then DELETE SUCC[s]
8.                      else begin s ← SUCC[s];
9.                                 η_j ← s
10.                            end;
11.                 j ← j + 1;
12.                 s ← SUCC[x]
        end
    end.
```

The action of the algorithm is illustrated in Fig. 5, where string segments are schematically shown as straight line segments and numbered consecutively $s_1, s_2, \ldots, s_7$.

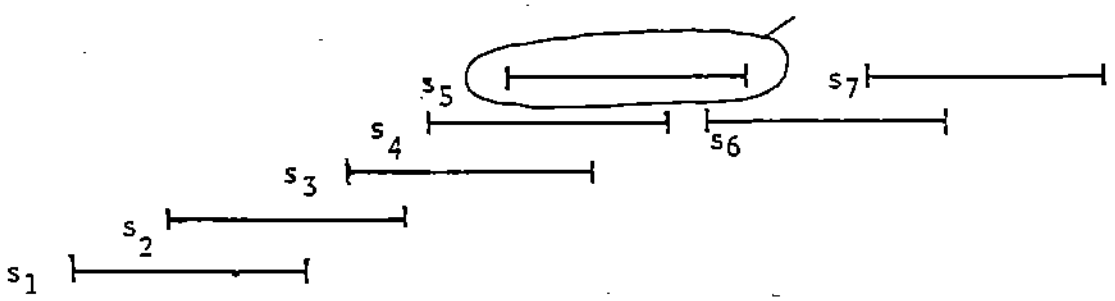

Figure 5
The extraction of necklaces from a run of segments.

Segment s_1 is entered into necklace η_1 by step 4, while segments s_2, s_3 and s_4 are entered by step 9. After entering s_4, the test in step 7 passes and segment s_5 is deleted. Next, the condition of the *while* loop (at line 6) no longer holds, since $s_6 = \mathrm{SUCC}[s_4]$ does not overlap s_4, and necklace η_2 is initialized in steps 11

and 12. Thus we extract the two necklaces η_1 and η_2 shown in Fig. 6.

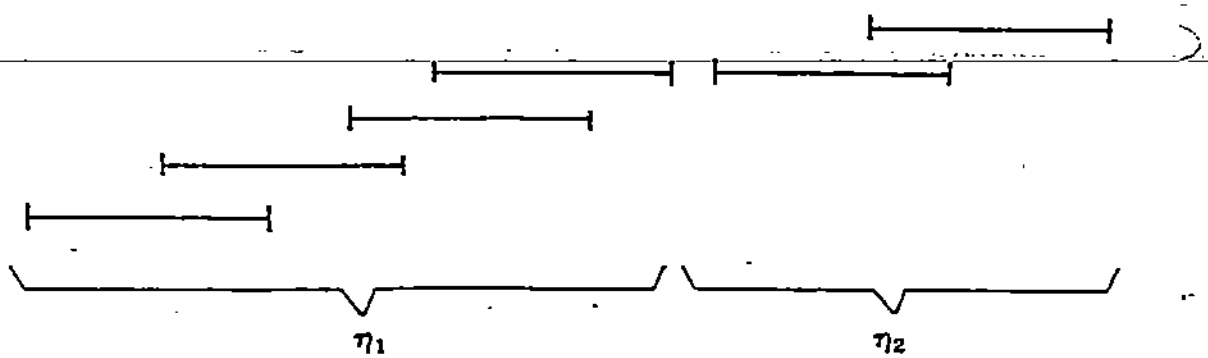

Figure 6
The two necklaces extracted from the run of Fig. 5.

For ease of reference, it will be convenient to consider each necklace as an alternating sequence of *master* and *slave* segments, the first term being a master. Also, a necklace is *odd* or *even* depending upon the parity of the number of its segments. (Both η_1 and η_2 above are even necklaces.)

Note that, by the definition of necklace, two consecutive segments of a necklace cannot be disjoint occurrences of $W(\alpha)$. However, since only consecutive segments overlap in a necklace, we have the following straightforward result.

Lemma 1: The value of $C_2(W(\alpha))$ equals the number of master segments in all necklaces at α.

It must be noted that there may be leaves of $S(\alpha)$ falling within the span of a necklace that do not belong to it. This may happen when the smallest period p of $W(\alpha)$ is less than $|W(\alpha)|/2$ and $W(\alpha) = (st)^k s'$ is a substring of some maximal repetition $R(i,p,L)$ of x in the form $(st)^k s$, with $k > k' \geq 2$ and $|s| > |s'|$. This

suggests consideration of a new type of subassembly in a run which is quite relevant to our objectives.

Definition 3 [GA]: A *chunk* is a maximal substring of a run such that $i_j - i_{j-1}$ $(j = 2,3,\ldots,s)$ is not larger than $p < |W(\alpha)|/2$.

As an immediate consequence of the periodicity lemma we have:

Lemma 2 [GA]: If $W(\alpha)$ has minimum period $p < |W(\alpha)|/2$, then the difference between the starting positions of two consecutive segments in a chunk is exactly p.

Notice that a given run may contain more than one chunk, as indicated in the example of Fig. 7.

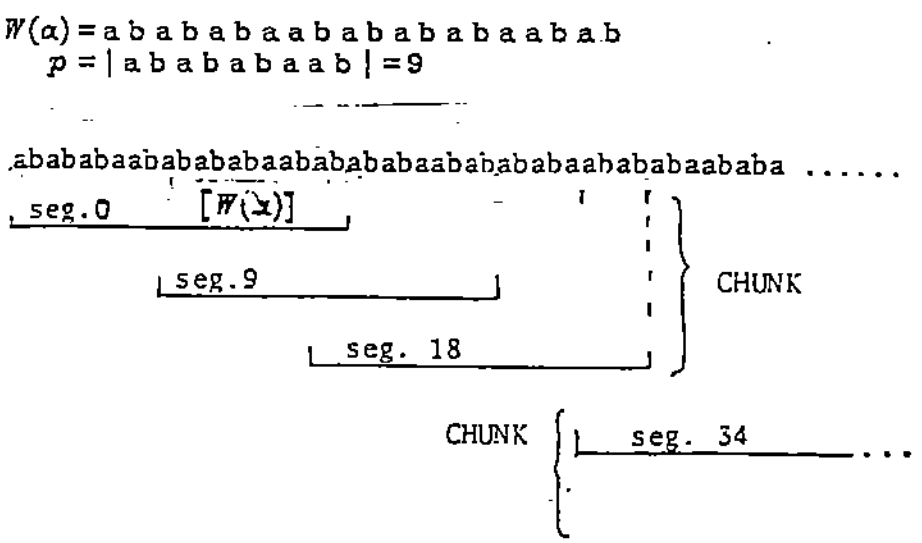

Figure 7
A run may contain more than one chunk.

As shown in this example, two consecutive chunks in the same run may overlap.

However, this overlap is bounded as prescribed by the following lemma:

Lemma 3: Two consecutive chunks at α may overlap at most for $q < p$ positions, where $p < |W(\alpha)|/2$ is the minimum period of $W(\alpha)$.

Proof:

Let $i_1, i_2, \ldots, i_m$ and $j_1, j_2 \ldots j_r$ be the segment in the first and second (consecutive) chunks, respectively. It follows from Lemma 2 that $j_1 > i_m$. We assume, for a contradiction, that $j_1 < i_m + W(\alpha) - p$. Notice that $j_1 \neq i_m + cp$ (for some $c = \lfloor |W(\alpha)| / p \rfloor$), otherwise, j_1 would belong to the same chunk as i_m.

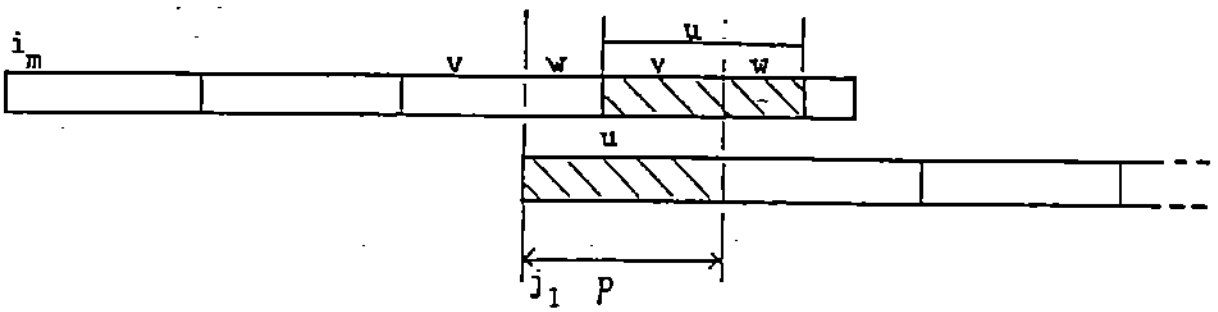

Figure 8

Assuming that two consecutive chunks at α overlap on more than $p-1$ positions generates a contradiction.

Letting u be the root of $W(\alpha)$, this means that there are words w and v such that it is $u = wv = vw$ (see Fig. 8), i.e., there is a non-trivial cyclic shift of u that turns this word into itself. But this implies that it is $u = v^t$ or $u = w^t$ with $t \geq 2$, which contradicts the assumption that u be primitive. □

For future reference, we remark that the contribution of a chunk to the C_2 value of the necklace to which it belongs may be retrieved at once from the knowledge of $|W(\alpha)|$, of the period p of $W(\alpha)$, of the span of the chunk and of the type (master or slave) of its first segment i_1, the chunk head. In fact, i_1 must be necessarily either a master or a slave segment by Lemma 3, and the same is

true of the second segment i_2 as well. If i_1 is in a chunk of span L, then there are exactly $l \triangleq 1+(L-|W(\alpha)|)/p$ segments in the chunk starting at positions i_1, $i_1+p, \ldots, i_1+lp$ by Lemma 2. Let i_j $(j=1,2)$ be the leftmost master segment in the chunk, and let $\tilde{C}_2$ be the contribution of the span of the chunk to $C_2(W(\alpha))$. Then tedious yet straightforward manipulations show that:

$$\tilde{C}_2 = \begin{cases} \left\lfloor \dfrac{L_j}{|W(\alpha)|} \right\rfloor & \text{if } W(\alpha)=u^k \text{ with } u \in I^+ \\ \left\lfloor \dfrac{L_j+|t|}{|W(\alpha)|+|t|} \right\rfloor & \text{if } W(\alpha)=(st)^k s \text{ with } s,t \in I^+ \end{cases}$$

with:

$$L_j = \begin{cases} L & \text{if } j=1 \\ L-p & \text{if } j=2 \end{cases}$$

We conclude this section with the following simple observation. For each chunk of $W(\alpha)$, we call *chunkhead* the above segment i_j $(j=1,2)$ and assume that the needed information (span, period, etc.) is available. Then the value of $C_2(W(\alpha))$ can be counted by identifying in $S(\alpha)$ the number of master segments that belong to a necklace, but not to a chunk, and then adding to the value thus obtained the contribution of each individual chunk.*

While this section considers the structural properties of runs, chunks and necklaces as static objects, the next section will consider the dynamical behavior of these objects when constructing $\hat{T}_x$ as a modification of T_x.

* We remark that, by the definition of necklace, if $i_1, i_2 \cdots i_k$ are consecutive segments w with $|w| > (i_j - i_{j-1})$, $j=1,2,\ldots,k-1$, then the construction of necklaces for all segments w of x partitions the set $I \equiv \{i_1, i_2 \cdots i_k\}$ into two subsets I_1 and I_2 (with I_2 possibly empty) so that the segments of I_1 are all in the same necklace and those in I_2 are deleted. In particular, the segments in a chunk cannot be partitioned between two distinct necklaces.

7. DYNAMIC RUNS, CHUNKS AND NECKLACES

We now revisit the bottom-up construction of $\hat{T}_x$ from T_x as outlined at the beginning of Section 5. For the sake of simplicity, we shall assume that T_x is initially a binary tree (this does not affect the generality of our discussion [AP]), and we will denote by $\overline{T}_x$ the partially updated structure that is obtained from T_x by the time the algorithm handles node α. Node α can be a node originally in T_x or an auxiliary node recently inserted. In this latter case it remains to be determined whether node α is in $\overline{T}_x$ to stay or must be dropped as a redundant unary node. Letting $S(\alpha)$, $S_L(\alpha)$ and $S_R(\alpha)$ (with one of the two latter possibly empty) be the necklace-structured sets of segments of length $|W(\alpha)|$ and pertaining, respectively, to the subtrees of $\overline{T}_x$ rooted at α, LSON(α), and RSON(α) (again, LSON or RSON may be empty), and assuming without loss of generality $|S_L(\alpha)| \geq |S_R(\alpha)|$, the task to be performed at α can be decomposed into the following subtasks:

1. (*MERGE*) - if both $S_R(\alpha)$ and $S_L(\alpha)$ are nonempty, then merge $S_R(\alpha)$ into $S_L(\alpha)$ by inserting one leaf at a time in succession, thus producing the necklace-structured set $S(\alpha)$ (as a byproduct of the merge, we get the value of $C_2(W(\alpha))$).

2. (*DELETE*) - else (α has only one son) if $C_2(W(\alpha)) = C_2(W(\text{SON}(\alpha)))$ then delete α; otherwise assign to α weight $C_2(W(\alpha))$.

3. (*CLIMB*) - Determine the node ν to be considered next as FATHER(α) and, if such node does not exist in $\overline{T}_x$, *create* it (this is the mechanism that inserts auxiliary nodes). Use $S(\alpha)$ to construct a necklace-structured set to be called $S_L(\nu)$ or $S_R(\nu)$ according to whether α = LSON(ν) or α = RSON(ν).

DELETE is a trivial subtask. In the hypothesis that both LSON(α) and RSON(α) exist, we first analyze the mechanism giving rise to the necklaces at α during the execution of MERGE.

MERGE

With the approach of inserting $S_R(\alpha)$ into $S_L(\alpha)$ leaf-by-leaf in sequence, the initial condition - i.e., prior to this merging operation - is given by the sequence of segments, of length $|W(\alpha)|$, in positions $\{i \mid i$ is a leaf of $S_L(\alpha).\}$

It is convenient to define Λ, the empty necklace, whose parity is trivially even (zero). It is an immediate consequence of Lemma 0 that no segment from $S_R(\alpha)$ being inserted may fall entirely within the span of a necklace in $S_L(\alpha)$. In fact no such segment may fall entirely within the span of a run of segments of $S_L(\alpha)$.

Thus each segment i (originating from a leaf of $S_R(\alpha)$) always bridges two necklaces, η_1 and η_2, both of which may be empty. The two necklaces η_1 and η_2 are concatenated by segment i into a new necklace η, which contains all the segments of η_1 and η_2, in addition to segment i. We also have: parity(η) = [parity(η_1) + parity(η_2) + 1] (mod. 2). Notice that all segments retain their original parity, except when parity(η_1)=0, in which case the masters of η_2 become slaves and *vice versa*. Finally, it is straightforward to prove the following:

Lemma 4: Segment i belongs to the compact $W(\alpha)$-tagging of x if and only if its insertion bridges two even necklaces of $S_L(\alpha)$.

Lemma 4, whose proof is left as an exercise, provides the desired criterion for the computation of $C_2(W(\alpha))$: assuming that the value of $C_2(W(\alpha))$ has been updated for all segments preceding segment i in S, then the latter contributes a unit increment to the count if it bridges two even necklaces of $S_L(\alpha)$.

As a final remark, we note that whenever segment i joins at least one nonempty necklace, then i is by definition the origin or the detector of a repetition of x. This latter might call for the insertion of auxiliary nodes in $\overline{T}_x$ at some later step; thus all the relevant information must be kept for possible later use.

CLIMB

The above remark is of use when analyzing the third subtask, CLIMB. As outlined before, the first step to be carried out here is the identification of FATHER(α). If $W(\alpha) = vv'$, with v' a prefix of v, $v^2 \in V_x$ and $|v| > |W(\text{FATHER}(\alpha)|$ in $\overline{T}_x$, then an auxiliary node is inserted in the proper locus of v and becomes FATHER(α) = ν. Otherwise FATHER(α) in $\overline{T}_x$ will coincide with FATHER(α) in T_x. The next task is to update the necklace-structured set $S(\alpha)$ to obtain $S(\nu)$. This process can be viewed as the contraction of each segment of $S(\alpha)$ to achieve its new length $|W(\nu)| < |W(\alpha)|$; however, in this process some overlaps may disappear, so that some necklaces of $S(\alpha)$ may break. The parities of the resulting fragments (which are necklaces of $S(\nu)$) have to be correctly set to obtain the desired $S(\nu)$.

We now concentrate on the mechanics of this segment contraction. We only need to examine the left-to-right sequence of the segments whose contraction cleaves a necklace of $S(\alpha)$.

Let then i be the generic such segment and assume that the necklace structure to the left of i has reached its final status. If i had a detector j in a necklace η of $S(\alpha)$, then η splits now in two nonempty necklaces η_1 and η_2, i becoming the last segment in η_1 and j the first segment of η_2. In addition, if j was a slave in η, then all slave segments in $\eta \cap \eta_2$ become master and *vice versa*. If (m,k) is the span of η and C_2', C_2'' are, respectively, the sizes of the compact $W(\nu)$-taggings of $x(1,k)$ before and after the contraction of i, this may result in

$C_2'' = C_2' + 1$, as the following example shows.

Example 3: Let $x = a\,b\,a\,a\,a\,b\,a\,a\,a\,b\,a\,a\,a\,b\,a\,b\,a\,a\,a\,b\,a\,a\,a\,b\,a\,a\,a\,b\,a$. Since $a\,b\,a\,a\,a\,b\,a\,a$ and $a\,b\,a\,a\,a\,b\,a\,b$ both are in V_x, then $a\,b\,a\,a\,a\,b\,a$ has a proper locus α in T_x. It is easy to see that W = a b a a a b has no proper locus in T_x. However, since W is also the root of a repetition of x, then W might require a proper locus ν in the MAST of x. In climbing from α to ν, all segments in the (unique) necklace η of $S(\alpha)$ undergo one unit contraction (refer to Fig. 9: master segments of η are shown solid; the bottom alignment displays the new tagging of x which results from the contraction of all segments of η).

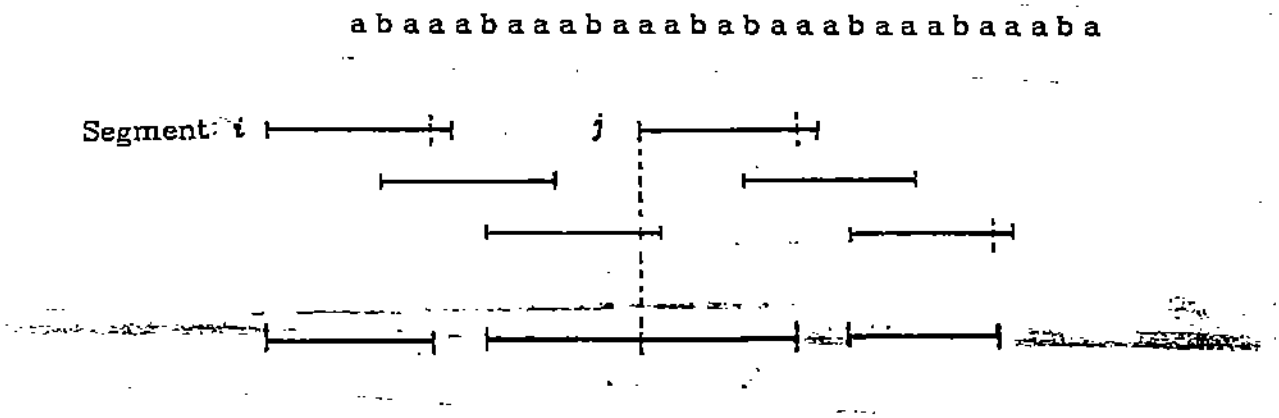

Figure 9
The effect of the contraction of segments in a necklace.

Scanning the segments of η from left to right, we see that, after the contraction of each of the first two segments of η, we have $C_2'' = C_2' = 3$. However, the contraction of segment i cleaves segment j, which was formerly a slave segment of η. In this particular example, the parity switching of all segments to the right of j brings the rightmost segment of η into the $W(\nu)$-tagging of x, whence $C_2'' = C_2' + 1 = 4$.

Lemma 5: $C_2'' = C_2' + 1$ if and only if j is a slave in η and η_2 is an odd necklace.

Proof:

If j is a master segment of η then the contraction of the (slave) segment i has no effect on the tagging and, obviously, $C_2'' = C_2'$. If j is a slave in η, and η_2 is an even necklace, then exchanging the role of master and slave leaves in $\eta \cap \eta_2$ will not augment the tagging since there are as many slaves in η_2 as there are masters. On

the other hand, if j is a slave in η and η_2 is odd, then the number of slave segments in $\eta \cap \eta_2$ exceeds by one that of master segments. Since the parity of such segments switch in the transition from η to η_2, an extra segment results in the tagging. □

6. CONCLUSION

A suitably augmented version of the suffix tree associated with a textstring

x is well suited to store the statistics without overlap of all substrings of x. The space required is shown here to be $O(n \log n)$. However, the existence of strings requiring $\vartheta(n \log n)$ space is still an open problem. The construction of the augmented suffix tree in its minimal form can be carried out almost straightforwardly in time $O(n^2)$. This paper was devoted to studying the structural properties of clusters of overlapping occurrences of the same substring in the text-string, as well as to analyzing the dynamic behavior of such clusters.

Apart from the intrinsic combinatorial interest of this investigation, the algorithmic criteria derived here can be exploited to set up a more efficient computation of the statistics without overlap, as we will show in a forthcoming paper.

9. REFERENCES

[AA] A. Apostolico, *On Context Constrained Squares and Repetitions in a String*, The R.A.I.R.O. Journal of Theoretical Informatics 18, 2, 147-159 (1984).

[AH] A. V. Aho, J.E. Hopcroft and J. D. Ullman, *The Design and Analysis of Computer Algorithms*, Addison Wesley, Ma. (1974).

[AG] A. Apostolico and R. Giancarlo, *The Boyer-Moore-Galil String Searching Strategy Revisited*, SIAM J. on Computing (to appear, 1985). An extended abstract appears in the Proceedings of the XX Allerton Conference on Communications, Control and Computing, Monticello, ILL. (1982).

[AL] A. Apostolico, *The Myriad Virtues of Subword Trees*, in: Combinatorial Algorithms on Words, (A. Apostolico and Z. Galil, eds.) Springer-Verlag (1985).

[AP] A. Apostolico and F. P. Preparata, *Optimal Off-line Detection of Repetitions in a String*, Theoretical Comp. Sci., 22, 297-315 (1983).

[BM] R. S. Boyer and J. S. Moore, *A Fast String Searching Algorithm*, Comm. of the ACM, 20, 323-350 (1977).

[CM] M. Crochemore, *Recherche Lineaire d'un Carre dans un Mot*, C. R. Acad. Sc. Paris, t.296, Serie I, 781-784 (1983).

[CR] M. Crochemore, *An Optimal Algorithm for Computing the Repetitions in a Word*, IPL 12,5, 244-250 (1981).

[GA] Z. Galil, *On Improving the Worst Case Running Time of the Boyer Moore String Matching Algorithm*, Proceedings of the ICALP (G. Ausiello and C. Boehm eds.), Springer-Verlag Lecture Notes in Computer Sciences, 62, 241-250 (1978).

[KM] D. E. Knuth, J. H. Morris and V. R. Pratt, *Fast Pattern Matching in Strings*, SIAM Journal on Computing, 6, 323-350 (1977).

[LE] A. Lentin and M.P. Schutzemberger, *A Combinatorial Problem in the Theory of Free Monoids*, Proc. of the University of North Caroline, 128-144 (1967).

[LM] R. J. Lorentz and M. G. Main, *Linear Time Recognition of Square-free Strings*, in: Combinatorial Algorithms on Words (A. Apostolico and Z. Galil, eds.), Springer-Verlag (1985).

[LO] M. Lothaire, *Combinatorics on Words*, Addison-Wesley, Ma. (1983).

[LS] R. C. Lyndon and M.P. Schutzemberger, *The Equation* $a^M = b^N c^P$ *in a Free Group*, Mich. Math. Journal, 9, 289-298 (1962).

[MC] E. M. McCreight, *A Space Economical Suffix Tree Construction Algorithm*, Jour. of the ACM 23, 262-272 (1976).

[ML] M. Main and R. Lorentz, *An* $O(n \log n)$ *Algorithm for Finding All Repetitions in a String*, The Journal of Algorithms, 422-432 (1984).

[MR] M. E. Majster and A. Reiser, *Efficient On-line Construction and Correction of Position Trees*, SIAM Journal on Computing, 9, 785-807 (1980).

[WE] P. Weiner, *Linear Pattern Matching Algorithms*, Proceedings of the 14th Annual Symposium on Switching and Automa Theory, (1973).