1993

Data Structures and Algorithms for the String Statistics Problem

Alberto Apostolico

Franco P. Preparata

Report Number:
93-085

DATA STRUCTURES AND ALGORITHMS FOR THE STRING STATISTICS PROBLEM

Alberto Apostolico
Franco P. Preparata

CSD-TR-93-085
December 1993

Data Structures and Algorithms for the String Statistics Problem *

Alberto Apostolico† Franco P. Preparata‡

December 15, 1993

Abstract

Given a textstring x of length n, the *Minimal Augmented Suffix Tree* $\hat{T}(x)$ of x is a digital-search index that returns, for any *query* string w and in a number of comparisons bounded by the length of w, the maximum number of nonoverlapping occurrences of w in x. It is shown that, denoting with n the length of x, $\hat{T}(x)$ can be built in time $O(n \log^2 n)$ and space $O(n \log n)$, off-line on a RAM.

Key Words and Phrases: Design and analysis of algorithms, combinatorics on strings, pattern matching, substring statistics, suffix tree, augmented suffix tree, period of a string, repetition in a string.

AMS subject classification: 68C25

---

*This research was supported in part, through the Leonardo Fibonacci Institute, by the Istituto Trentino di Cultura, Trento Italy.

†Department of Computer Science, Purdue University, West Lafayette, IN 47907, USA and Dipartimento di Elettronica e Informatica, Università di Padova, Padova, Italy, axa@cs.purdue.edu. Additional support was provided by NSF Grants CCR-8900305 and CCR-9201078, by NATO Grant CRG 900293, and by the National Research Council of Italy.

‡Department of Computer Science, Brown University, Providence, R.I., USA, franco@cs.brown.edu. Additional support was provided by NSF Grant CCR-91-96176 and ONR Contract N 00014-91-J-4052, ARPA order 2225.

1 Preliminaries

Let x be a *string* (*word*) of lenght $|x| = n$ on an alphabet I. Denoting by w a substring of x, $C(w)$ is the maximum number of distinct *nonoverlapping* occurrences of w in x. For example, $w = aba$ occurs 11 times in $x = abaababaabaabababaababababaa$, with starting positions in the set $\{1, 4, 6, 9, 12, 14, 17, 19, 21, 23, 25\}$ (cf. Fig. 1). However, some such occurrences overlap with each other, e.g., those starting at positions 4 and 6, or 12 and 14, etc. We can choose up to 7 occurrences of w in x so that no two of them overlap, e.g., those with starting positions in $\{1, 4, 9, 12, 17, 21, 25\}$. Thus, $C(aba) = 7$.

In this paper, we address the construction of a digital-search index designed to report, for any substring w of x and in $O(|w|)$ comparisons, the value of $C(w)$. The basic structure of such an index, introduced in [AP-85], is the suffix tree of string x augmented with a cardinality value $c(\alpha)$ at each internal node α. Integer $c(\alpha)$ gives the number of occurrences without overlap in x of the string w associated with α. One such tree is illustrated in Figures 2a and 2b. Recall that the *suffix tree* $T(x)$ of a string x is a $(|I| + 1)$-ary tree ($|I|$ being the alphabet size) where each leaf corresponds to a string position, edges are labeled with (pairs of pointers to positions of x that identify) substrings of x. A root-to-leaf path describes in a natural way the suffix of x beginning at the position associated with the leaf. Moreover, any substring w of x is associated either with a node or with an edge of the tree (called the *locus* of w). When seeking *all* occurrences (with overlap), the number of occurrences of a substring w is trivially given by the number of leaves reachable from the locus of w: thus, to obtain this statistics, it is sufficient to label each internal node α with the number $\bar{c}(\alpha)$ of the leaves in the subtree rooted at α. Our present objective is instead to augment the suffix tree $T(x)$, so that all relevant loci are labeled with the correct cardinalities (see Figure 2b). Specifically: each internal node α is labeled with an integer $c(\alpha) \leq \bar{c}(\alpha)$ and new (unary) *auxiliary* nodes are inserted along edges whenever a cardinality change occurs. The resulting structure is the Minimal Augmented Suffix Tree $\hat{T}(x)$ of x. Without loss of generality, we assume for simplicity $|I| = 2$.

The c-labeling of the suffix tree $T(x)$ can be viewed as a "pebbling" process. The standard pebbling policy in a rooted tree is that leaves can be unconditionally pebbled and that an internal node can be pebbled only when all

of its children have already been pebbled. Pebbling a node α means to produce a data structure $\Lambda(\alpha)$ containing the relevant information about the substring whose locus is α. From this data structure $\Lambda(\alpha)$ we can compute the parameter $c(\alpha)$ in a straightforward way. We denote $L(\alpha)$ and $R(\alpha)$ the two children (left and right, resp.) of α. Assuming that $\Lambda(L(\alpha))$ and $\Lambda(R(\alpha))$ are available, $\Lambda(\alpha)$ is constructed as follows:

1. Process the edges leading from $L(\alpha)$ (resp., $R(\alpha)$) to α, inserting all possible auxiliary nodes corresponding to changes of cardinality (operation $CLIMB$).
2. Merge the data structures obtained after subjecting $\Lambda(L(\alpha))$ and $\Lambda(R(\alpha))$ to operation $CLIMB$ (operation $MERGE$).

Figuratively, $CLIMB$ and $MERGE$ process edges and nodes, respectively.

2 The Data Structures

As mentioned above, a node α in $T(x)$ is the locus of some string w, and each leaf of the subtree $\tau(\alpha)$ of $T(x)$ rooted at α identifies an occurrence of w in x; any such leaf is denoted by an integer i giving the position in x of the leftmost symbol in the occurrence of w.

Two occurrences i_1 and i_2 $(i_2 > i_1)$ of w (w-occurrences) are said to *overlap* if $i_2 - i_1 < |w|$. A *necklace* (of w-occurrences) is a maximal sequence $\mathcal{N} = \{i_1, i_2, ..., i_h\}$ of w-occurrences such that no other w-occurrence $i, (i < i_1)$ overlaps with i_1 and only consecutive terms in $\mathcal{N}$ overlap. The contribution of a necklace in $\tau(\alpha)$ to $c(\alpha)$ is obviously the ceiling of half of the necklace size. Thus, knowing the necklaces in $\tau(\alpha)$ is all that is needed to compute $c(\alpha)$. Data structure $\Lambda(\alpha)$ is designed to store the necklaces of $\tau(\alpha)$.

Structure $\Lambda(\alpha)$ is a two-level tree, such that the leaves of the upper-level are the necklaces, and each necklace is in turn represented by a tree whose leaves are w-occurrences. It must support operations *INSERT, DELETE, SPLIT, SPLICE, FIND, RANK* (here $RANK(i)$ is the ordinal number of i in the necklace), and is therefore a minor modification of a standard concatenable queue [AHU-74].

As we shall see in detail below, pebbling a node involves merging two analogous structures by successively inserting the terms of the smaller one into the larger one; any such insertion may extend or concatenate necklaces, but never splits existing ones. However, pebbling an edge is a more subtle operation. As we (figuratively) proceed rootward along an edge, the string length decreases and two formerly *overlapping* w-occurrences i_1 and i_2 may become disjoint w'-occurrences, where w' is a prefix of w. This happens because x contains a repeated substring in the form $w'w'$ starting at position i_1. It is therefore essential to devise a technique designed to process these events, referred to as *jumps*. Let $string(\alpha)$ denote the string whose locus is α. Recall that an integer p is a *period* for a string w if $w_i = w_{i+p}, i = 1, 2, |w| - p$ (note that a string may have more than one period). A jump on edge $(CHILD(\alpha), \alpha)$ occurs if and only if the string associated with some descendant of $CHILD(\alpha)$ has the form $w'w'$ where the period $p = |w'|$ is in the range $[|string(CHILD(\alpha))| - 1, |string(\alpha)|]$. This condition is easily tested if we keep an ordered list (as a dictionary) of the periods so far discovered in the pebbling of the subtree $\tau(\alpha)$ of $T(x)$ rooted at α: indeed, a period is detected each time we encounter two overlapping w-occurrences. To efficiently process all occurrences associated with a given period p, it is convenient that the record for p contain a pointer to a secondary data structure (itself a dictionary) storing, as an ordered list $thread(p)$, all such occurrences. Clearly, each such occurrence will cause a necklace split in some structure Λ.

In conclusion, we keep two types of data structures associated with each "pebble".

The first, called *P-directory*, has, as a primary component, an ordered list of periods (integers), each of which points to a secondary component (*P-thread*) storing the corresponding occurrences (see Figure 3). Primary and secondary structures are dictionaries.

The second type of data structure is the Λ structure described earlier.

It is clear that these two structures closely interact during the construction of the index. Specifically, each time a w-occurrence overlap is detected during a $MERGE$ operation, an appropriate modification is introduced in the P-directory. Conversely, each time during a $CLIMB$ operation a period p is detected within the length range of an edge, then the P-directory is used to effect the necessary necklace splits in the Λ-structure.

We now outline the basic structure of the operations $MERGE$ and $CLIMB$ under the simplifying assumption that the w-occurrences involved refer to a substring w of x such that w is not *periodic*. A string w is periodic if w has a period $p \leq \lfloor |w|/2 \rfloor$. In our construction, the treatment of periodic substrings is more involved, and is deferred to Section 4.

3 The Case of Nonperiodic Strings

3.1 The Operation MERGE

The operation $MERGE$ constructs $\Lambda(\alpha)$ from the analogous data structures $\{\Lambda^*(\beta) : \beta \text{ a child of } \alpha\}$, where $\Lambda^*(\beta)$ is the result of processing $\Lambda(\beta)$ through the procedure $CLIMB$ applied to edge (β, α).

Assuming $I = \{a, b\}$, $\Lambda^*(L(\alpha))$ and $\Lambda^*(R(\alpha))$ respectively contain all necklaces of $string(\alpha)$-occurrences followed by a and b. If, without loss of generality, we have $|\Lambda^*(L(\alpha))| \geq |\Lambda^*(R(\alpha))|$, then $\Lambda(\alpha)$ is obtained by successively inserting each leaf (a $string(\alpha)$-occurrence) of $\Lambda^*(R(\alpha))$ into $\Lambda^*(L(\alpha))$. Let j be the position of $\Lambda^*(R(\alpha))$ being inserted into $\Lambda^*(L(\alpha))$, and let it fall in the interval $[i_1, i_2]$ of $\Lambda^*(L(\alpha))$. Denoting $w = string(\alpha)$, we have the following three cases:

(i) $j - i_1 \geq |w|$ and $i_2 - j \geq |w|$: j is a new necklace.

(ii) $j - i_1 < |w|$ or $i_2 - j < |w|$, but not both: j is appended (at the beginning or at the end) to an existing necklace.

(iii) $j - i_1 < |w|$ and $i_2 - j < |w|$: j causes two existing necklaces to merge into a single necklace.

In each one of these three cases, the *parity* of the resulting necklace is updated in a straightforward fashion as the ceiling of one half the $RANK$ of the last element of that necklace. In cases (ii) and (iii) we encounter overlapping occurrences. Any time $j - i_1 < |w|$, period $p = j - i_1$ is detected; period p is inserted into the primary component of the P-directory (if not already present), and j is inserted into the list $thread(p)$ pointed to by p. Analogously for $i_2 - j < |w|$.

3.2 The Operation CLIMB

The operation $CLIMB$ inserts all necessary auxiliary nodes within an edge of $T(x)$, with their c statistics, and produces the Λ^* data structure ready for the merging operation.

Let (α_1, α_2) be the current edge, where α_1 is a child of α_2, and let $w_i = \mathit{string}(\alpha_i), i = 1, 2$. Moreover, let $|w_2| < p_1 < p_2 < ... < p_s < |w_1|$ be the sequence of periods internal to the interval $[|w_2|, |w_1|]$ in the appropriate P-directory: up to s nodes might have to be inserted within the edge, each the locus of a sequence of length $p_i, i = 1, 2, ..., s$.

The sequence $(p_1, p_2, ..., p_s)$ is obtained by a search in the primary component of the associated P-directory.

If $s > 0$, then we successively process $p_s, p_{s-1}, ..., p_1$. While processing p_i, we access the secondary component $\mathit{thread}(p_i)$ and visit all its members (a sequence of positions). Each such position corresponds to a necklace split in the Λ- structure being processed (the number of necklaces increases, although no new occurrence is created). This may or may not result in an increase in the c statistics with respect to the last node pebbled along edge (α_1, α_2): in case such a cardinality change occurred, a new node is inserted and weighted appropriately. After this operation, list $\mathit{thread}(p_i)$ becomes useless. We stress that $CLIMB$ inserts a new node only when this is warranted by a change in the c statistics, whence the final augmented suffix tree is *minimal*.

Finally, it might happen that $|w_2|$ is also a period in the P-directory. If this is the case, the current $CLIMB$ must process also $\mathit{thread}(|w_2|)$ in order to produce one of the Λ^* structures needed for the $MERGE$ at α_2. Processing of $\mathit{thread}(|w_2|)$ is is not different from the other periods, except that the locus α_2 of w_2 now already exists (and cannot be deleted). Clearly, if the sequence of periods p_i is empty ($s = 0$) and $|w_2|$ is not found in the primary component of the P-directory, then $CLIMB$ is void and $\Lambda^*(\alpha_1) = \Lambda(\alpha_1)$.

4 The General Case: Periodic and Nonperiodic Strings

The procedure described is adequate if w is a nonperiodic string. As announced at the end of Section 2, we consider now the general case, which includes nonperiodic as well as periodic strings.

Recall that a string w is periodic if it has a period $p \leq \lfloor |w|/2 \rfloor$. The main reason for the insuitability of the previous procedure to periodic strings is that for a periodic w it is no longer necessarily the case that every w-occurrence belongs to some necklace (see Fig. 4). Thus, if we maintain unmodified the notion of necklaces of w-occurrences, it is not hard to see that an update (insertion) may *obliterate* a w-occurrence. As it turns out, when dealing with periodic strings, the notion of "necklace occurrence" is no longer convenient, and is replaced by the more appropriate notion of "chunk", defined below, which reflects the high regularity of periodic strings.

A well-known fact of combinatorics on words, referred to as the *periodicity lemma* [LS62], states that if a string w has two periods $p, q \leq \lfloor |w|/2 \rfloor$, then also $g.c.d(p,q)$ is a period of w. An easy consequence of the periodicity lemma is that given a *maximal* run of w-occurrences $\{i_1, i_2, ...i_s\}$ such that $i_j - i_{j-1} \leq |w|/2$ $(j = 2, 3, ..., s)$ then $i_j - i_{j-1} = p$ $(j = 2, 3, ..., s)$, where p is the *minimum* period of w.

This allows us to replace, for periodic strings, the notion of w-occurrence with the notion of "chunk": a *w-chunk* is a run $(i, i+p, \ldots, i+(s-1)p)$ of w-occurrences, $p \leq \lfloor |w|/2 \rfloor$ (cf. Fig. 4); the *span* L of this chunk is equal to $|w| + (s-1)p$ (the length of the text segment from the initial position of the first w-occurrence to the terminal one of the last w-occurrence).

During the periodic regime (i.e., while $|w| > 2p$) the following holds [AP85]: two consecutive chunks may overlap in at most $p-1$ positions. Thus, rather than with necklaces of w-occurrences, we shall deal with necklaces of w-chunks (of course an isolated w-chunk is a degenerate necklace).

The w-occurrences of a w-chunk obey the following strong constraint. For $|w| > 2p$ and for any three w-occurrences $i_1 < i_2 < i_3$ the lowest common ascendant $lca(i_1, i_2)$ of i_1 and i_2 in the suffix tree $T(x)$ is a *descendant* of $lca(i_2, i_3)$. Indeed, let $w = u^r u'$, where u is a primitive string, $r \geq 2$ and u' is a prefix of u. Then the suffix of the text x beginning at i_j is $u^{((i_3 - i_j)/|u|)} u^r u' t$

for some string t and $j = 1, 2, 3$. Clearly, the suffixes pertaining to leaves i_1 and i_2 share a prefix whose length exceeds by $i_3 - i_2$ that of the prefix shared by the suffixes pertaining to i_2 and i_3. This implies that the path from the root to $lca(i_1, i_2)$ contains $lca(i_2, i_3)$.

This constraint has a few significant consequences:

(i) For a w-chunk of period p starting at i, we successively encounter (as we climb the rootward path from leaf i of $T(x)$) the lca of w-occurrence i and w-occurrences $i + p, i + 2p, ...$

(ii) A w-chunk is detected (and established) when its second w-occurrence at $i + p$ is detected. When the chunk is established, its span is $L = |w| + p$.

(iii) As we climb the path of $T(x)$, up from the node where the chunk is established, we scan prefixes w' of w. As long as $|w'| \geq 2p$, the span assumes periodically the values $L, L - 1, ..., L - p + 1$. Therefore if two chunks overlap (which occurs when the leftmost one achieves its maximum span) as the span length contracts the two chunks become disjoint (since their overlap is strictly less than p) and this happens exactly once during each period.

(iv) The cardinality s of w'-occurrences steps up by one unit exactly when the span attains its maximum value.

A simple analysis shows that an isolated w-chunk (i.e., a chunk not overlapping with any other chunk) contributes

$$c = \lfloor \lceil L/p \rceil / \lceil |w|/p \rceil \rfloor \qquad (1)$$

units to the c-statistics. Indeed, in the set of w-occurrences associated with the chunk each disjoint w-occurrence uses an integral number of periods, for a total length of $\lceil |w|/p \rceil p$. Referring to the *extended span* $\lceil L/p \rceil p$, also extended to a length corresponding to an integral number of periods, it is immediate that c is given by the above formula. This value of c is referred to as the chunk's *nominal contribution.*

It is convenient to define, for each w-chunk, an additional structural parameter ξ, called *excess*, given by

$$\xi = \lceil L/p \rceil mod \lceil |w|/p \rceil \qquad (2)$$

which specifies the number of periods, within the chunk span, that are not utilized in the chunk's (nominal) contribution to the c-statistics. We claim that the actual contribution of a chunk to the c-statistics is always given by (1), except in the following extremely special situation:

a) the chunk belongs to a nondegenerate necklace;

b) the chunk has $\xi = 0$;

c) the chunk is preceded, in its necklace, by a maximal string of chunks whose excesses form a $\{0,1\}$-string with an odd number of zeroes.

Indeed, in this situation the leftmost period of the chunk is not utilizable (since it is used by the preceding chunk), and since the chunk excess is 0, its actual contribution to the c-statistics is one less than its nominal contribution. The three conditions $a), b)$ and $c)$ above will be collectively referred to as Condition A.

Therefore, for a given w, a w-chunk is specified by a triplet (i,p,s), where i is the initial position (*chunkhead*), p the period and s the number of w-occurrences within the chunk; ξ is readily computable from these parameters.

We are now ready to describe our adaptations of $MERGE$ and $CLIMB$ in greater detail, adopting the above-defined triplet notation for chunks. As usual, a necklace of chunks is structured as a height-balanced binary tree, whose leaves are the chunks and whose internal nodes contain parameters (to be introduced below) instrumental in the $MERGE$ and $CLIMB$ processes.

4.1 Adapting MERGE

We consider $MERGE$ first, and concentrate as earlier on the insertion of an individual leaf from $\Lambda^*(R(\alpha))$ into $\Lambda^*(L(\alpha))$.

Notice that whenever we merge the two sets of occurrences of a chunk $(i_1, i_2, ..., i_s)$, originating respectively in $\Lambda^*(R(\alpha))$ and $\Lambda^*(L(\alpha))$, these two sets have necessarily the forms $(i_1, i_2, ..., i_{s-1})$ and (i_s). Indeed, by the hypothesized maximum span condition, as the cardinality of occurrences increases by one, the alphabet symbol following occurrence i_s is necessarily

different from the (common) one following occurrences $i_1, i_2, ..., i_{s-1}$. Therefore, occurrences $\{i_1, i_2, ..., i_{s-1}\}$ and $\{i_s\}$ originate from distinct children of node α (the locus of $w = string(\alpha)$). Occurrence i_s is referred to as the *tail* of the chunk.

Suppose now that i is the item of $\Lambda^*(R(\alpha))$ being inserted into $\Lambda^*(L(\alpha))$, and let i fall in the interval $[l, r]$ of (the current version of) $\Lambda^*(L(\alpha))$. In light of the above discussion, the following three cases are possible.

(a) i is the tail of a chunk of $\Lambda(\alpha)$, of which l is the head (see Fig. 5.a).

(b) i is the head of a chunk of $\Lambda(\alpha)$, of which r is the tail (Fig. 5.b).

(c) i is the head of a (possibly degenerate) chunk that $\Lambda(\alpha)$ inherits entirely from $\Lambda^*(R(\alpha))$ (Fig. 5.c).

Straightforward calculations, based on i and on the triplets associated with l and r enable us to decide which one of cases (a), (b) or (c) applies to i. We examine case (a) in some detail; handling of the other cases is analogous. Let $\mathcal{N}_1$ and $\mathcal{N}_2$ be the (consecutive) necklaces containing l and r, respectively, each represented by a search tree $t(\mathcal{N}_i)$ whose leaves are chunks.

Occurrence i belongs to the rightmost chunk (l, p, s) of $\mathcal{N}_1$ (the rightmost leaf of $t(\mathcal{N}_1)$). Chunk (l, p, s) must be updated to $(l, p, s+1)$. Let ξ be the excess of $(l, p, s+1)$. If $\xi = 0$, then we must check whether Part c) of Condition A holds, and, if so, whether the contribution of $\mathcal{N}_2$ is affected.

This can be done with the help of two ternary parameters $(\eta_L(\nu), \eta_R(\nu))$ assigned to each node ν of a necklace tree. These parameters are defined as:

ν is a leaf $\qquad \eta_L(\nu) = \max(\xi, 2)$

ν is internal $\quad \begin{cases} \eta_L(\nu) = \eta_L(\text{leftchild}(\nu)) * \eta_L(\text{rightchild}(\nu)) \\ \eta_R(\nu) = \eta_R(\text{rightchild}(\nu)) * \eta_R(\text{leftchild}(\nu)) \end{cases}$

where operation * has the following table

*	0	1	2
0	1	0	2
1	0	1	2
2	0	2	2

It can be easily verified that: $\eta_L(\nu) = 0$ if and only if Part c) of Condition A holds for a maximal suffix of the leaves in the subtree of ν; $\eta_L(\nu) = 1$ if and only if, the subtree v behaves as an excess string $11\ldots1$; $\eta_L(\nu) = 2$, otherwise. Therefore, traversing the rightmost path of $t(\mathcal{N}_1)$, we explore the sequence of η_L parameters of the roots of this path's left subtrees: if the maximal $\{0,1\}$-suffix of this sequence is $\ldots01\ldots1$, then Part c) of Condition A holds for chunk $(l, p, s+1)$. Therefore, if also $\xi = 0$, the actual contribution of $(l, p, s+1)$ is one unit less than its nominal one, since $|w| > 2p$, the rightmost period of $(l, p, s+1)$ is not utilized and the contribution of $\mathcal{N}_2$ is unaffected, even if $\mathcal{N}_2$ overlaps with $\mathcal{N}_1$.

On the other hand, if Condition A, Part (c) holds and $\xi = 1$, then the rightmost period of $(l, p, s+1)$ is utilized, and this may affect the contribution of $\mathcal{N}_2$ (if it overlaps with $\mathcal{N}_1$). A little reflection will establish that the η_R parameter of the root of $t(\mathcal{N}_2)$ has value 0 if and only if there is a maximal prefix of chunks with {0,1} excesses containing an odd number of 0-excess chunks: in such a case the effect of the overlap is to reduce by one unit the nominal contribution of $\mathcal{N}_2$.

Finally, if overlap occurs, $\mathcal{N}_1$ and $\mathcal{N}_2$ must be merged by splicing their trees, using standard techniques and adjusting the η_L, η_R parameters as well.

We now describe the necessary modifications of the P-directory. Let $p_1 < p_2 < \ldots < p_t$ be the periods currently stored in the primary component of the P-directory. Assuming that $string(\alpha)$ is periodic, then p_1 is the *only* period $\leq \lfloor |w|/2 \rfloor$, and $thread(p_1)$ contains the ordered list of the chunkheads; therefore, referring to the case described above, l is to be inserted if and only if $i - l = p$, that is, when the chunk is first established. On the other hand, all of the other periods correspond to chunk overlaps, i.e., $|w| - p < p < |w|, j = 2, \ldots, t$; therefore, if i overlaps with r, then i is to be unconditionally inserted into $thread(r - i)$. Note that i is not explicitly represented in the necklace, but a pointer to l (the chunk-head of i) will do. List $thread(r - i)$ will be used in the ensuing $CLIMB$ operation.

Handling of cases (b) and (c) is analogous.

4.2 Adapting CLIMB

Recall the rationale for a *CLIMB*: as we proceed along an edge toward the root, the string length decreases and two w-occurrences i_1 and i_2 that

formerly overlapped may become disjoint w'-occurrences, where w' is a prefix of w. We are interested in such events, earlier called "jumps", since they induce changes in the necklace structure, which in turn might affect the c statistics. Before addressing the mechanics of such changes, we must show that all jumps are detected by our algorithm.

Handling of nonperiodic w (i.e., when w is not of the form $w = v^k v'$, for $k > 1$ and some prefix v' of v) has been discussed in Section 4. The characteristic condition of a jump is string $w'w'$ (a square) starting at i_1: we must ensure that the list $thread(|w'|)$ contains item i_1. But it is known [AP83] that there are nodes α in $T(x)$ with $|\text{ string }(\alpha)| \geq |w'|$ where i_1 and i_2 are consecutive leaves in $\tau(\alpha)$; at one such node i_1 and i_2 join in a MERGE operation, which adjoins i_1 to $thread(|i_2 - i_1|)$.

We now consider the case of periodic $w = v^k v$, and let w' denote a prefix of w. As $|w'|$ decreases, the following events may occur, which may affect the c-statistics.

Event 1. The nominal contribution of a chunk may increase. This is due to the fact that $\lfloor \lceil L/p \rceil / \lceil |w'|/p \rceil \rfloor > \lfloor \lceil (L+1)/p \rceil / \lceil (|w'|+1)/p \rceil \rfloor$ and may occur only when p divides $|w'|$, in which case both L/p and $|w'|/p$ are integers. For this chunk, such event occurs at most once in a period of the span length.

Event 2. Overlapping chunks j_1 and j_2 become disjoint (corresponding to a split of a chunk necklace). This, of course, does not affect the nominal chunk contributions, which are exclusively determined by Event 1 above. Again, for this pair of chunks, such event occurs at most once in a period.

The occurrence of Event 1 along the edge $(\alpha, PARENT(\alpha))$ in $T(x)$ can be easily predicted on the basis of parameters L, p, and $|w|$ (when the chunk's s-parameter steps up), and, if so, the value $|w'| = kp$ for which this will happen can be determined. A possible way to handle this situation is to insert an appropriate record into $thread(|w'|)$, although technically $|w'|$ is *not* a period, as defined earlier. Another type of record will be inserted into $thread(|w'|)$ for $|w'| = 2p$ for each chunk of period p, indicating the transition from periodic to aperiodic regime, and therefore the dissolution of the chunks.

Therefore, there are three basic actions which may occur when procedure $CLIMB$ is applied to arc $(\alpha, PARENT(\alpha))$. Denoting by $h_1 < ... < h_r$ the items in the primary component of the P-directory (such that $thread(h_j)$ is

nonempty), with $h_1 \geq |string(PARENT(\alpha))|$ and $h_r < |string(\alpha)|$, these actions are concisely summarized as follows:

```
begin for  i = r down to  1 do
    if p is not a divisor of h_i then process Event 2
    else
        if  h_i ≠ 2p then process Event 1
        else  dissolve chunks
end.
```

If j is associated with Event 1 (which means that p divides $|w'|$), then, traversing the path from the root of $t(\mathcal{N})$ to leaf j, by a straightforward modification of the argument developed in Section 4.1 we can determine the *effective* span length L' of the chunk, i.e., whether L or $L-p$ is to be utilized in computing the chunk's contribution, and compute the latter.

If j is associated with Event 2, then $\mathcal{N}$ splits into necklaces $\mathcal{N}_1$ and $\mathcal{N}_2$. Binary search for j in $t(\mathcal{N})$ defines the splitting path. On the basis of the (η_L, η_R) parameters observed on this path, we can establish the contributions of the separated chunks.

When the length of the string $|w'|$ becomes $2p$, we change from the periodic to the nonperiodic regime illustrated in Section 3.1 and 3.2, where again we deal with necklaces of occurrences, and all occurrences are explicit. To achieve this situation, we must generate the sequence of w'-occurrences resulting from the fragmentation of the pre-existing chunks (here w' is the prefix of w of length $2p$).

If we know the set of chunks $\{(i_j, p, s_j) : j = 1, 2, ..., m\}$, we can easily generate explicit w'-occurrences for each chunk (we have w'-occurrences at $i_h + kp$, $h = 1, 2, \ldots, m$ and $k = 0, 1, \ldots, s_j - 1$). Note that local knowledge of the rank of each chunk head enables us to assign individual ranks and parities to the explicit w'-occurrences that now replace that chunk.

5 Perfomance Analysis

Let $\hat{T}_x$ denote the structure to be produced by our construction. As already said, we can produce $\hat{T}_x$ from string x in two phases: in Phase 1 we build $T(x)$ and in Phase 2 we transform it into $\hat{T}_x$ by carrying out the pebbling process described in this paper. Phase 1 (building $T(x)$ from x) requires time $O(n \log |I|)$ and linear space, where n is the length of x and $|I| \leq n$ is the size of the alphabet [Mc-76]. We prove below that Phase 2 requires $O(n \log^2 n)$ time and $O(n \log n)$ space. Thus, the overall cost of building $\hat{T}_x$ from x is $O(n \log^2 n)$ time and $O(n \log n)$ space.

Consider first the space. Structure $\hat{T}_x$, the *minimal augmented suffix tree*, has the property that the number of auxiliary nodes inserted in $T(x)$ is minimum (every auxiliary node in $\hat{T}_x$ has a c-statistics different from that of its child node). As proved in [AP-85], each auxiliary node can be injectively charged to a square in x. Since the number of squares in a string of length n is bounded by $O(n \log n)$, then this bound translates to the total number of auxiliary nodes, whence to the total space taken by $\hat{T}_x$.

We claim that, at any given time, the space required by the collection of all "pebbles" is $O(n)$, and is therefore dominated by the size of $\hat{T}(x)$. Indeed, as illustrated in Section 2, each abstract "pebble" is concretely represented by two data structures, the Λ-structure and the P-directory. Each of these two structures is implemented as a two-level balanced tree, so that their space requirement is linear in the cardinalites of their respective sets of leaves. Note that each leaf appears at most once in either type of data structure, and, at any time, the set of leaves of all "pebbles" is a subset of the set of leaves of $T(x)$. Since $T(x)$ has n leaves, the claim follows.

Next, consider the time complexity. We begin by analyzing $MERGE$. The discussion in Section 3.1 shows that each Λ-structure insertion uses a bounded number of elementary operations from our expanded repertoire of concatenable-queue primitives, and therefore takes $O(\log n)$ time. Moreover, each $MERGE$ is carried out by inserting a smaller set within a larger set, so that any given item is never inserted more than $\log n$ times. In conclusion, the total time taken by the $MERGE$ is $O(n \log^2 n)$.

We now turn to procedure $CLIMB$. We recall that $CLIMB$ processes an edge according to one of three possible events: a necklace split occurring in the periodic regime (referred to as Event 2 in Section 4.2), a cardinality step-

up due to string contraction, and the dissolution of a chunk into conventional necklaces.

Beginning with Event 2, we observe that the necklace split is doable in time $O(\log n)$. Since each occurrence of Event 2 is injectively associated with a square $w'w'$, as a substring of x, and the total number of squares in x is $O(n \log n)$, handling of Event 2 takes time $O(n \log^2 n)$.

Each cardinality step-up, either in the periodic or the nonperiodic regime, is also injectively associated with a unique square $w'w'$. Again, since the total number of squares is $O(n \log n)$, we conclude that handling of cardinality step-ups runs in time $O(n \log^2 n)$.

Finally, we consider the time required to dissolve all chunks of period p, which are listed in *thread*$(2p)$ (see Section 4.2). For each chunk (i, p, s), the procedure generates explicit w-occurrences $i + kp$, $(k = 0, 1, \ldots, s - 1)$, and inserts each one appropriately into the necklace that originally contained only i, at a cost of $O(\log n)$ time. Note, however, that $i+kp$ $(k = 0, 1, \ldots, s-1)$ is the starting position in x of a square of period p. Again, we charge the generation and insertion of w-occurrence $i + kp$ to the corresponding square $(k = 0, 1, \ldots, s - 1)$. Since each square is charged only once in connection with chunk dissolution, the total number of such charges is bounded by $O(n \log n)$, for a total time of $O(n \log^2 n)$.

We conclude with the following statement:

Theorem. The construction of the Minimal Augmented Suffix Tree for a string of length n can be accomplished in time $O(n \log^2 n)$ with space $O(n \log n)$.

6 References

AHU-74 A. V. Aho, J. E. Hopcroft and J. D. Ullman, *The Design and Analysis of Computer Algorithms*, Addison Wesley, Ma. (1974).

Ap-85 A. Apostolico, The Myriad Virtues of Subword Trees, in *Combinatorial Algorithms on Words* (A. Apostolico and Z. Galil, eds.), Springer-Verlag ASI F-12, 85-95 (1985).

AP-83 A. Apostolico and F. P. Preparata, Optimal Off-line Detection of Repetitions in a String, *Theoretical Computer Science*, **22**,

297-515 (1983).

AP-85 A. APOSTOLICO AND F.P. PREPARATA, Structural Properties of the String Statistics Problem, *Journal of Computer and System Sciences*, **31**, 3, 394-411 (1985).

LS-62 R. C. LYNDON AND M. P. SCHUTZENBERGER, The Equation $a^M = b^N c^P$ in a Free Group, *Mich. Math. Journal* **9**, 289-298 (1962).

Mc-76 E. M. MCCREIGHT, A Space Economical Suffix Tree Construction Algorithm, *Jour. of the ACM*, **25**, 262-272 (1976).

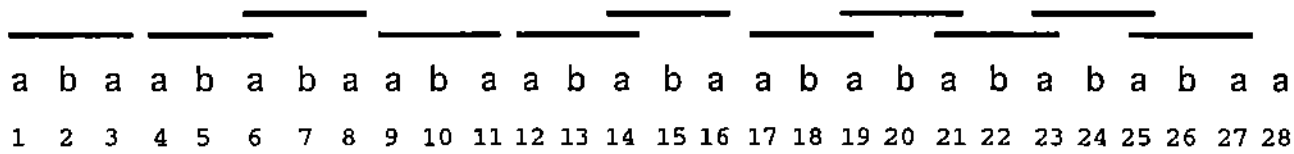


Figure 1

The string $w = aba$ has 11 occurrences in $x = abaababaabaababaababababaa$, but no more than 7 such occurrences are mutually disjoint.

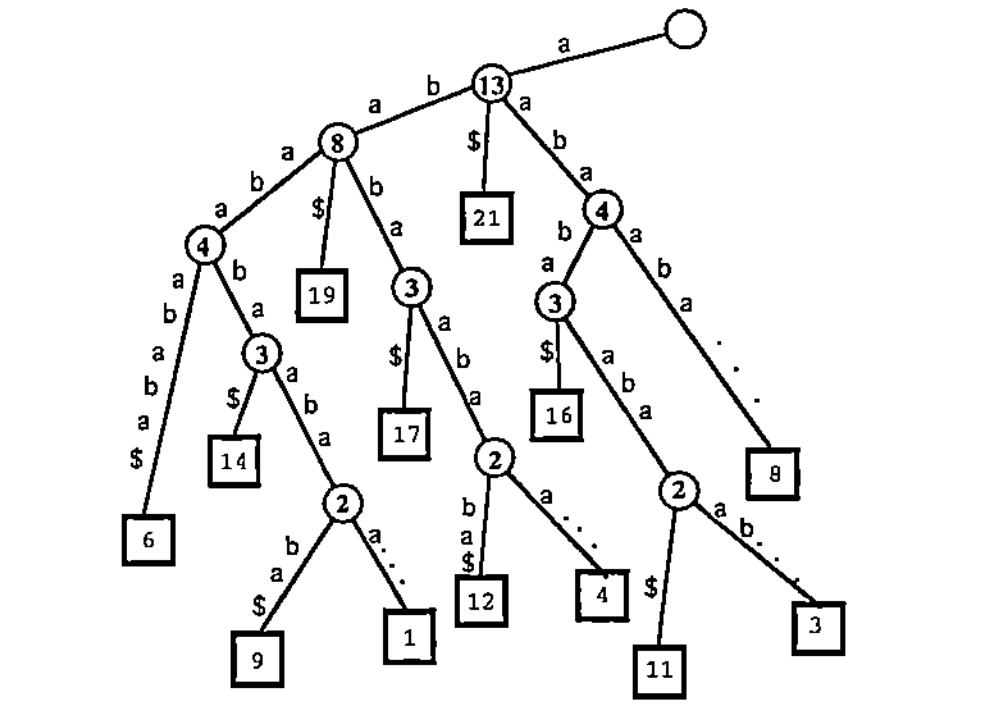
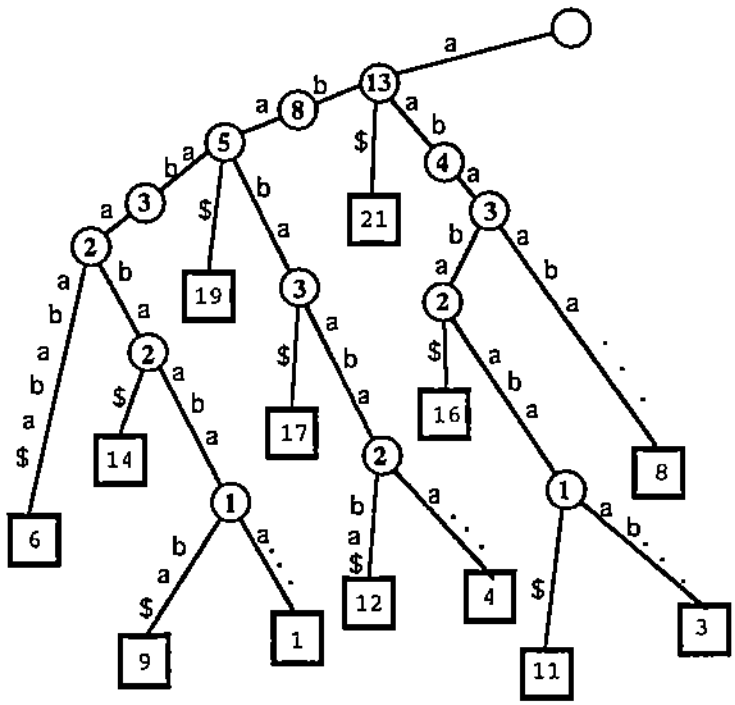


Figure 2.a

A partial view (all suffixes starting with ``a'') of the weighted suffix tree for the string $x = abaababaabaababaababa$: the weight of each internal node reports the number of (possibly overlapping) occurrences in x of the substring having locus at that node. Symbol $\$$ is a special marker not appearing elsewhere in x. To achieve linear space, the substring labeling each arc is encoded in practice by a pair of pointers to the beginning and end of one of its occurrences in x.



1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
a	b	a	a	b	a	b	a	a	b	a	a	b	a	b	a	a	b	a	b	a	$

Figure 2.b

Partial view of the MAST for the string of Figure 2.a.: the weights of most internal nodes now reflect the statistics without overlaps, and new nodes had to be inserted to account for changes in such a statistics that occur while ``climbing" along some of the arcs.

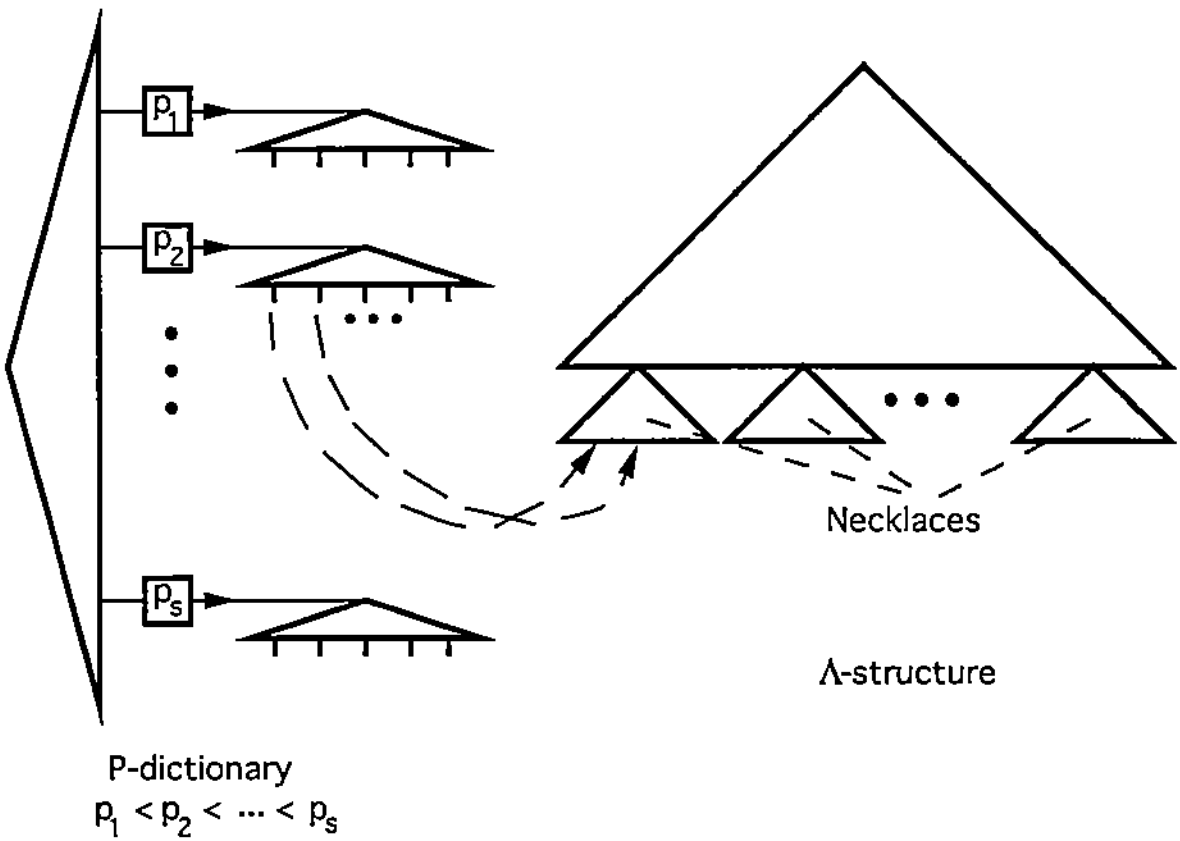


Figure 3

Salient data structures in a pebble for nonperiodic substrings: the Λ-structure collects necklaces of substring occurrences, while the P-dictionary provides access to threads of overlapping occurrences relative to a same period (only two pointers from the p_2 -thead are shown).

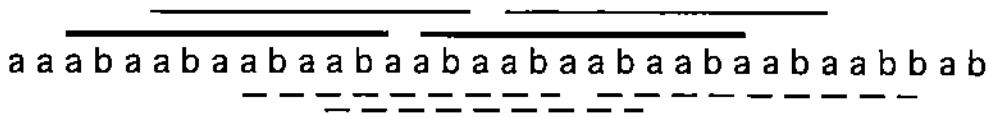


Figure 4

Greedy extraction of necklace segments from an isolated run of occurrences. Necklace occurrences are solid, with bold lines denoting odd occurrences. The dashed segments represent spurious occurrences.

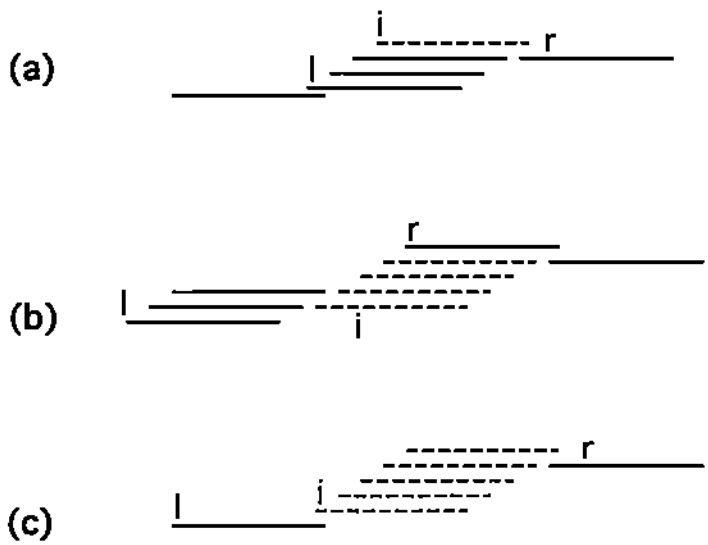


Figure 5

The three basic cases of merging with chunks; solid segments represent occurrences already in place, and the occurrences in each newcomer chunk are dashed.