# SELF-ALIGNMENTS IN WORDS AND THEIR APPLICATIONS*

Alberto Apostolico† Wojciech Szpankowski‡

**Abstract**

Some quantities associated with periodicities in words are analyzed within the *Bernoulli* probabilistic model. In particular, the following problem is addressed. Assume that a string $X$ is given, with symbols emitted randomly but independently according to some known distribution of probabilities. Then, for each pair $(W, Z)$ of distinct suffixes of $X$, the expected length of the longest common prefix of $W$ and $Z$ is sought. The collection of these lengths, that are called here *self-alignments*, plays a crucial role in several algorithmic problems on words, such as building suffix trees or inverted files, detecting squares and other regularities, computing substring statistics, etc. The asymptotically best algorithms for these problems are quite complex and thus risk to be unpractical. The present analysis of self-alignments and related measures suggests that, in a variety of cases, more straightforward algorithmic solutions may yield comparable or even better performances.

**Key words and phrases**: Combinatorial algorithms on words, average case analysis of algorithms, Bernoulli model, self-alignments, periodicities in words, suffix trees, substring statistics.

*This work was supported in part by the NSF Grant CCR-8900305, by AFOSR Grant 90-0107, and by Grant R01 LM05118 from the National Library of Medicine.

‡Department of Computer Science, Purdue University, West Lafayette, IN 47907, USA, and Dipartimento di Matematica Pura e Applicata, University of L'Aquila, Italy. Additional support for this author was provided by the NATO Grant CRG 900293, by the Italian Ministry of Research and by the National Research Council of Italy.

‡Department of Computer Science, Purdue University, West Lafayette, IN 47907, USA. Additional support for this author was provided by NSF Grant NCR-8702115 and by NATO Grant CRG 890057.

## 1. INTRODUCTION

Periodicities and related phenomena in words are known to play a central role in many facets of theoretical computer science, notably, in coding theory, in the theory of formal languages and in the design and analysis of algorithms. In this latter field, several efficient algorithms have been set up to date both to detect and to exploit the presence of repeated subpatterns and other kinds of avoidable or unavoidable regularities in words [LO]. In this paper, we focus on a class of algorithmic problems that share the following common feature. The efficiency with which these problems can be solved depends in a crucial way on the time taken to answer, once or repeatedly, the following basic question: given a word $X$, and two arbitrary suffixes $W$ and $Z$ of $X$, what is the (or length of the) longest common prefix of $W$ and $Z$? Some of these problems have met already solutions that are optimal in the asymptotic sense. For others, the asymptotically best available solutions are either suboptimal or not known to be optimal. In all cases, the associated constructions are generally quite elegant, but also quite involved. As is often the case, the main complications arise from the need to cope efficiently with worst-case inputs, no matter how pathological and thus unrealistic such inputs look like. By contrast, straightforward constructions also exist that do not present, in asymptotic terms, unbearable computational overheads with respect to the more elaborate ones. Moreover, the asymptotic time complexities of such simpler constructions typically hide smaller constants. Thus, simpler algorithms might be equivalent or even faster in practice.

In this paper, we consider random strings formed by symbols that are drawn independently from some finite alphabet, according to any fixed distribution of probabilities. We find that, for such strings, the algorithmic problems considered can be solved by straightforward methods within *expected* asymptotic time complexities that are for some problems only slightly worse, and for other problems equal to or even better than the *worst-case* asymptotic time complexities that affect the corresponding clever methods.

The paper is organized as follows. In Section 2, we introduce some measures of correlation among the subwords of a word. (We find it convenient to assume our words unbounded, but the upper bounds that we derive based on this assumption will hold *a fortiori* for words of finite length.) In particular, given any pair of distinct suffixes of an unbounded random word $X$, we derive the distribution function of the length of the longest common prefix of those suffixes. Using this, we prove that the maximum such length over all pairs that use only the first $n$ suffixes of $X$ has expected value $O(\log n)$. Moreover, the same bound holds for the expected value of the average, over the first $n$ suffixes, of the length of the longest prefix of each suffix that matches a previous prefix. In Section 3, we apply our probabilistic results to the average case analysis of some simple algorithmic solutions of important problems on words. We summarize the main results of that section, referring to the case of a binary input string of $n$ symbols emitted by a symmetric source. We find that building the suffix tree for such a word, which takes linear time by clever methods [MC], takes $O(n \log n)$ time by the direct method; detecting all squares in that word, which takes optimal $O(n \log n)$ time by clever methods [AP, CR, ML], takes $O(n \log n)$ expected time by a simpler method; computing the full statistics without overlap of all substrings of that word, which takes $O(n \log^2 n)$ time by clever methods [AP1, AP2], takes $O(n \log n)$ expected time by a simpler method, etc. The same asymptotic bounds hold in the case of nonuniform distributions, although the constants involved grow with the sum of squares of probabilities associated with a source symbol. Section 4 concludes our discussion by relating the present results to those obtained by previous studies on general tries [FL, KN, PI, SZ1-SZ3].

## 2. AUTOCORRELATION PARAMETERS IN WORDS

In this section, we introduce some basic definitions and present an analysis of self-alignments of a word in a probabilistic framework.

### 2.1 Basic Definitions and Summary of Main Results

Let $X = x_1 x_2 x_3 \ldots$ be a string of unbounded length formed by symbols from an alphabet $\Sigma$ of cardinality $V$, and let $S_i = x_i x_{i+1} \ldots$ be the $i$-th *suffix* of $X$, $i = 1, 2, \ldots.$ For every off-diagonal pair $(i, j)$ of positions of $X$, we define $C_{ij}$ as the length of the longest string that is a prefix of both $S_i$ and $S_j$. We leave $C_{ij}$ undefined when $i = j$. Thus, $C_{ij} = k$ iff $i \neq j$ and $S_i$ and $S_j$ agree exactly on their first $k$ symbols, but differ on their $(k+1)$st. Clearly, $C_{ij} = C_{ji}$ for all meaningful choices of $i$ and $j$.

Let now $n$ be any fixed integer. The following expressions define, in succession, the $n$-th *depth of the* $i$-th *suffix*, denoted $D_n(i)$, the $n$-th *height* $H_n$ of $X$, the $n$-th *shortest path* or *shallowness* $h_n$ of $X$, and the *typical* (or *average*) *depth* $D_n$ of $X$.

$$D_n(i) = \max_{1 \leq j \leq n, j \neq i} \{C_{ij}\} , \tag{2.1a}$$

$$H_n = \max_{1 \leq i < j \leq n} \{C_{ij}\} = \max_{1 \leq i \leq n} \{D_n(i)\} , \tag{2.1b}$$

$$h_n = \min_{1 \leq i \leq n} \max_{1 \leq j \leq n, j \neq i} \{C_{ij}\} = \min_{1 \leq i \leq n} \{D_n(i)\} , \tag{2.1c}$$

$$D_n = \frac{1}{n} \sum_{i=1}^{n} D_n(i) \tag{2.1d}$$

Intuitively, $H_n$ is the maximum possible length for a substring $Z$ of $X$ that has at least two occurrences in $X$, both starting within the first $n$ positions of $X$. Thus, there are two positions $i$ and $j$ of $X$, $i < j < n$, such that the occurrence of $Z$ starting at $j$ can be fully recopied from the occurrence starting at $i$. The typical depth $D_n$ represents the average length, over the first $n$ positions of $X$, of the longest substring of $X$ that can be recopied from the past. The height $H_n$ and its companion parameters express structural correlations among the substrings of string $X$. Such correlations play a crucial role in many combinatorial and algorithmic constructions, and our three definitions above are somewhat reminiscent of notions already appeared in the literature, notably, in [LZ, GO].

For a given $n$, the (symmetric) table collecting all meaningful values $C_{ij}$ is the $n$-th *self-alignment matrix* of $X$. In the following, we refer to a generic off-diagonal entry of this matrix by one of the terms *self-alignment* or *common* , the latter term being mnemonic for "length of the longest prefix *common* to a generic pair of suffixes of $X$". The following example illustrates the notions introduced so far.

EXAMPLE 2.1. *Illustrating definitions*

Let $X = abbabaa...$ and $n = 5$. Then $S_1 = X$, $S_2 = bbabaa...$, $S_3 = babaa...$, $S_4 = abaa...$ and $S_5 = baa...$ . The corresponding self-alignment matrix $\mathbf{C} = \{C_{ij}\}_{i,j=1}^{5}$ is as follows.

$$\mathbf{C} = \begin{bmatrix} \star & 0 & 0 & 2 & 0 \\ 0 & \star & 1 & 0 & 1 \\ 0 & 1 & \star & 0 & 2 \\ 2 & 0 & 0 & \star & 0 \\ 0 & 1 & 2 & 0 & \star \end{bmatrix}$$

From $\mathbf{C}$ and the expressions (2.1), we obtain $H_n = 2$, $h_n = 1$, and the average depth $D_n = (2 + 1 + 2 + 2 + 2)/5 = 9/5$. ■

We deal here with the probabilistic analysis of the above quantities under the *Bernoulli* assumptions: *the symbols of $X$ are drawn independently from $\Sigma$, and the i-th symbol of $\Sigma$ occurs in $X$ with probability $p_i$, and $\sum_{i=1}^{V} p_i = 1$.* We first compute the distributions of all random variables $C_{ij}$ $(1 \leq i, j \leq n)$ in the Bernoulli model, and then use such distributions to estimate the average values $EH_n$, $ED_n$ and $Eh_n$, however, in this paper we mainly concentrate on the evaluation of the height $H_n$. Towards this end, observe that our assumptions (notably, the unboundedness of $X$) entail that the distributions of $C_{ij}$ vary with $i$ and $j$ in a way that depends on the differences $d = |j - i|$ rather than on the specific individual values of $i$ and $j$. In other words, all random variables $C_{ij}$ having the same value of $d = |j - i|$ have the same distribution. Thus, it is appropriate to reason in terms of the random variables $C_d$, where $d = 1,\ 2\ ,...,\ n-1$. For example, $C_{1,2},\ C_{2,3}\ ,...,\ C_{n-1,n}$ have the same distribution, and are thus clustered in the new random variable $C_1$ (i.e., $d = 1$). We remark, however, that the random variables in a family such as $C_{1,d+1},\ C_{2,d+2}\ ,...,\ C_{n-d,n}$ are *dependent.*

Our main results of this section are summarized in the following proposition. Proof is given in the next two subsections.

**PROPOSITION** (i) *Let $d$ be any finite integer smaller than $n$, and let $l$ and $r$ be the unique integers defined by $k = dl + r$, where $l = 0,\ 1,\ ...,$ and $r < d$. Then,*

$$\Pr\{C_d = k\} = Pr\{C_d = ld+r\} = \left(\sum_{i=1}^{V} p_i^{l+2}\right)^r \left(\sum_{i=1}^{V} p_i^{l+1}(1-p_i)\right) \left(\sum_{i=1}^{V} p_i^{l+1}\right)^{d-r-1} . \qquad (2.2a)$$

*For the symmetric distribution $p_1 = p_2 = ... = p_V = 1/V$, expression (2.2a) simplifies to*

$$\Pr\{C_d = k\} = V^{-k}(1 - 1/V)\ . \qquad (2.2b)$$

*Finally, the complementary function to the distribution of $C_d$, denoted $R_d(k) = \Pr\{C_d \geq k\}$ becomes*

$$R_d(k) = \left(\sum_{i=1}^{V} p_i^{l+2}\right)^r \left(\sum_{i=1}^{V} p_i^{l+1}\right)^{d-r} . \qquad (2.3)$$

(ii) *The $n$-th average height $EH_n$ satisfies*

$$EH_n \leq a_n + C \qquad (2.4a)$$

*with*

$$a_n \sim \frac{2}{\log P^{-1}} \log n\ , \qquad (2.4b)$$

*where $C$ is a constant, and $P = \sum_{i=1}^{V} p_i^2$ is the probability that the symbols of two assigned distinct positions of a word match. Furthermore, $H_n \leq -2\log_P n$ in probability, that is, for every $\varepsilon > 0$ the following holds*

$$\lim_{n\to\infty} \Pr\{H_n \leq -2(1+\varepsilon)\log_P n\} = 1\ . \qquad (2.5)$$

*Finally, all the other parameters defined in (2.1), that is, the shortest path $h_n$ and the depth $D_n$, are $O(\log n)$.* ∎

Table 1
SIMULATION RESULTS OF SUFFIX TREES
*Asymmetric alphabet $p = 0.1$ and $q = 0.9$*

| $n$ | $-2\log_P n$ | $EH_n$ | $\frac{1}{E}\log n$ | $ED_n$ |
|---|---|---|---|---|
| 700 | 66.022 | 52.889 | 20.152 | 23.861 |
| 800 | 67.368 | 53.519 | 20.583 | 24.340 |
| 900 | 68.555 | 54.850 | 20.563 | 24.766 |
| 1000 | 69.617 | 56.119 | 21.249 | 25.080 |
| 2000 | 76.602 | 63.762 | 23.381 | 27.457 |
| 3000 | 80.688 | 67.930 | 24.629 | 28.802 |
| 4000 | 83.588 | 70.294 | 25.514 | 29.652 |
| 5000 | 85.837 | 73.153 | 26.200 | 30.348 |
| 10000 | 92.822 | 80.314 | 28.332 | 32.574 |

**Remarks**

(i) The evaluation of the distributions of the $C_d$'s is crucial for the rest of the paper, and in particular, for computing $EH_n$. Formula (2.2a) is surprisingly simple in the light of the strong dependencies between $S_i$ and $S_{i+d}$. More in general, we observe that $\Pr\{C_d = k\}$ does not depend on the fine structure of string $X$.

(ii) We conjecture that $EH_n \sim (2/\log P^{-1})\log n$, that is, the upper bound proved in our Proposition can be matched by a lower bound. This conjecture is a result of a simulation study that we have performed on a random string of symbols, and which is summarized in Table 1. The table indicates that $EH_n \sim \frac{2}{\log P^{-1}}\log n$, hence our upper bound seems asymptotically tight. [1]

Table 1 also shows the average depth $ED_n$ simulated in our experiment, and its comparison with $\frac{1}{E}\log n$ where $E$ is the *entropy* of the alphabet defined as $E = -\sum_{i=1}^{V} p_i \log p_i$. Again, the table suggests that asymptotically $ED_n \sim \frac{1}{E}\log n$. [2]

(iii) In practice, we are interested in finite strings in the form $X\$ = x_1x_2...x_n\$$, where \$ is a symbol not in the alphabet $\Sigma$. Even though the suffixes of $X$ are now finite, upper bound

[1] This conjecture was recently settled in the affirmative [DSR].

[2] This was recently proved in [JS] (see the last section of this paper for additional comments).

(2.4) in our Proposition still holds in these cases, since the height of a finite string *cannot* exceed the height of an unbounded string as assumed in our Proposition. ∎

**2.2 Probability distribution function of $C_d$**

In this subsection we prove part (i) of our Proposition, that is, we compute the probability distribution function $\Pr\{C_d = k\}$ of $C_d$ for $d = 1, 2, ..., n-1$. To simplify notations, we assume a binary alphabet (i.e., $V = 2$), but it will be understood that our derivations extend trivially to any finite alphabet. We set $p = p_1$ and $q = p_2 = 1 - p$, with obvious meaning.

The following known fact of combinatorics on words (cf., e.g., [LO]) plays a crucial role in our discussion.

**Fact 1**. *Let $(S_i, S_j)$ be any pair of suffixes of $X$ such that $i < j$ and $j - i = d$, let $Z$ be the longest common prefix of $S_i$ and $S_j$, and let $|Z| = C_{ij} = k \geq 0$. Then, string $Z$ can be written as $Z = U^\ell U'$, where $|U| = d$, $|U'| = r < d$, $U'$ is a prefix of $U$, and $U^\ell$ is the string resulting from the concatenation of $\ell$ copies of $U$.* ∎

The following example illustrates the probabilistic implications of Fact 1.

EXAMPLE 2.2. *Computing* $\Pr\{C_3 = k\}$ *for* $d = 3$ *and* $V = 2$

To fix the ideas, we consider $S_1$ and $S_4$. To enhance visual impact, we write $S_4 = y_1 y_2 y_3 ...$, so that the alignment of $S_1$ and $S_4$ is represented as in Fig.1.

$$\begin{array}{lllllllll} S_1 & x_1 & x_2 & x_3 & y_1 & y_2 & y_3 & y_4 & y_5 \\ S_4 & y_1 & y_2 & y_3 & y_4 & y_5 & y_6 & y_7 & y_8 \end{array}$$

Figure 1.

Fact 1 enables us to limit consideration the following three cases.

**CASE 1**: $k \leq d - 1 = 2$.

Let $P = p^2 + q^2$. Then, by our main probabilistic assumption, we immediately obtain

$$\begin{aligned} \Pr\{C_3 = 0\} &= \Pr\{x_1 \neq y_1\} = 1 - P \\ \Pr\{C_3 = 1\} &= \Pr\{x_1 = y_1, x_2 \neq y_2\} = P(1 - P) \\ \Pr\{C_3 = 2\} &= \Pr\{x_1 = y_1, x_2 = y_2, x_3 \neq y_3\} = P^2(1 - P) \end{aligned}$$

**CASE 2**: $3 = d \leq k \leq 2d - 1 = 5$.

We look first at $\Pr\{C_3 = 3\} = \Pr\{x_1 = y_1, x_2 = y_2,\ x_3 = y_3,\ y_4 \neq y_1\}$. Note that the events $\{x_1 = y_1\}$ and $\{y_4 \neq y_1\}$ are dependent, while $\{x_2 = y_2\}$ and $\{x_1 \ = y_3\}$ are independent. We proceed as follows.

$$\begin{aligned}
\Pr\{C_3 = 3\} &= \Pr\{x_1 = y_1 \neq y_4, x_2 = y_2, x_3 = y_3\} \\
&= \Pr\{x_1 = y_1 \neq y_4\}\Pr\{x_2 = y_2\}\Pr\{x_3 = y_3\} \\
&= (\Pr\{x_1 = y_1 \neq y_4 | y_1 = 0\} \cdot \Pr\{y_1 = 0\} + \Pr\{x_1 = y_1 \neq y_4 | y_1 = 1\} \cdot \Pr\{y_1 = 1\}) \cdot P^2 \\
&= (p^2 q + q^2 p) P^2
\end{aligned}$$

The crucial passage is in the second line of the above, where we replace the joint distribution $\Pr\{x_1 = y_1 \neq y_4 x_2 = y_2, x_3 = y_3\}$ by a product of probabilities of independent events. Along the same lines, we have

$$\begin{aligned}
\Pr\{C_3 = 4\} &= \Pr\{x_1 = y_1 = y_4 x_2 = y_2 \neq y_5 x_3 = y_3\} \\
&= \Pr\{x_1 = y_1 = y_4\} \cdot \Pr\{x_2 = y_2 \neq y_5\}\Pr\{x_3 = y_3\} \\
&= (p^3 + q^3)(p^2 q + q^2 p) P
\end{aligned}$$

and

$$\Pr\{C_3 = 5\} = (p^3 + q^3)(p^2 q + q^2 p) P^0$$

Note that, in this case, we have a factor $P$ in our probability. This factor disappears for $k \geq 2d$.

**CASE 3** $k \geq 2d = 6$.

In this case, we know from Fact 1 that $Z = U^l U'$ with $|U| = d = 3$, $|U'| = r$, $r < 3$ and $|Z| = d \cdot l + r$. Thus, we can group all symbols of $Z$ into $d = 3$ independent clusters and compute separately the probabilities in each group. For example, for $r = 1$, we have

$$\begin{aligned}
\Pr\{C_3 = 3l + 1\} &= \Pr\{x_1 = y_1 = ... = y_{3l+1}, x_2 = y_2 = ... \neq y_{3l+2}, x_3 = y_3 = ... = y_{3l}\} \\
&= (p^{l+2} + q^{l+2})(p^{l+1} q + q^{l+1} p)(p^{l+1} + q^{l+1})
\end{aligned}$$

In summary, in Case 1 all symbols are independent by our main assumption, so the probability is easy to evaluate. In Case 2 we have some symbols which appear twice, hence dependency starts playing a role in computing the probability. Finally, for $k \geq 2d$, the dependency is strong, but also disciplined by an entirely predictable structure. ■

In general, we can write $k = dl + r$ in a unique way. Fact 1 enables now to distribute the $k$ symbols of a common among $d$ mutually independent groups, and the value of $r$ identifies the group where an inequality holds. Hence

$$\Pr\{C_d = dl{+}r\} = \Pr\{x_1 = y_1 = ... = y_{dl+1}, ..., x_{r+1} = y_{r+1} = ... \neq y_{dl+r+1}, ..., x_d = ... = y_{dl}\}$$

$$= (p^{l+2} + q^{l+2})^r (p^{l+1}q + q^{l+1}p)(p^{l+1} + q^{l+1})^{d-r-1} \ . \qquad (2.6a)$$

In particular, for $l = 0$ we have:

$$\Pr\{C_d = r\} = \Pr\{x_1 = y_1,\ x_2 = y_2\ , ...,\ x_{r+1} \neq y_{r+1}\} = P^r(1-P) \qquad (2.6b)$$

where $P = p^2 + q^2$. For $l = 1$, we obtain:

$$\Pr\{C_d = l + r\} = (p^3 + q^3)^r (p^2 q + q^2 p) P^{d-r-1} \qquad (2.6c)$$

Generalization to alphabets of arbitrary size is straightforward, whence formula (2.2a) of Proposition (i). For the symmetric case, simple substitution of $1/V$ for the symbol probabilities of (2.2a) yields (2.2b).

Finally, to prove our formula (2.3) for the function $R_d(k) = \Pr\{C_d \geq k\}$ it suffices to carry out the above analysis with one small change. Namely, in this case suffixes, say $S_i$ and $S_{i+d}$, must agree on the first $k$ symbols, and we do *not* require that they differ on the $k + 1$st symbol.

### 2.3 The Height of Suffix Tree

In this subsection, we prove Proposition (ii). To compute the average height we need to evaluate the average value of the maximum of some dependent random variables. For the exact computation of such a maximum, we would need the joint distribution of all $C_{ij}$, $i, j = 1,\ 2\ , ...,\ n$. For our purposes, however, a good upper bound is sufficient. We shall derive such a bound on the basis of our knowledge of $\Pr\{C_d = k\}$ alone, using the following slight generalization of ideas already in [LR1, LR2] (cf. also [SZ2]).

**Lemma 1**. *Let $Y_1,\ Y_2\ , ...,\ Y_m$ be a sequence of random variables with distribution functions $F_1(y),\ F_2(y)\ , ...,\ F_m(y)$, respectively. Let $R_i(y) = \Pr\{Y_i \geq y\}$ be the complement function of the distribution function $F_i(y)$ (function $R(\cdot)$ is sometimes called the reliability function). Finally, let $\overline{M}_m = \max_{1 \leq i \leq m}\{Y_i\}$ and $\underline{M}_m = \min_{1 \leq i \leq m}\{Y_i\}$. Then:*

(i) *If $a_m$ is the smallest solution of*

$$\sum_{k=1}^{m} R_k(a_m) = 1 \ , \qquad (2.7)$$

*then*

$$E\overline{M}_m \leq a_m + \sum_{k=1}^{m} \sum_{j=a_m}^{\infty} R_k(j) \ . \qquad (2.8)$$

*(ii) If $b_m$ is the largest solution of*

$$\sum_{k=1}^{m} F_k(b_m) = 1 \ , \tag{2.9}$$

*then*

$$E\underline{M}_m \geq b_m - \sum_{k=1}^{m} \sum_{j=0}^{b_m} F_k(j) \ . \tag{2.10}$$

*(iii) If distribution functions $F_i(y)$ of $Y_1,\ Y_2\ ,...,\ Y_m$ satisfying for all $1 \leq i \leq m$ the following conditions*

$$F_i(y) < 1 \qquad \text{for all} \quad y < \infty \ , \tag{2.11}$$

*and*

$$\lim_{y\to\infty} \sup_i \frac{1 - F_i(cy)}{1 - F_i(y)} = 0 \qquad \text{for} \quad c > 1 \ , \tag{2.12}$$

*then $M_m/a_m \leq 1$ in probability, that is, for any $\varepsilon > 0$*

$$\lim_{m\to\infty} \Pr\{M_m \geq (1+\varepsilon)a_m\} = 0 \ , \tag{2.13}$$

*where $a_m$ solves asymptotically (2.7).*

**Proof**. For (i) observe that, for any $a$ (cf. [LR1]),

$$\overline{M}_m \leq a + \sum_{k=1}^{m} [Y_k - a]^+$$

where $t^+$ denotes $\max\{0,\ t\}$. Since $[Y_k - a]^+$ is a nonnegative random variable, then [FE] $E[Y_k - a]^+ = \int_a^\infty R_k(y)dy$, so that, (assuming for simplicity that $Y_i$ is a continuous random variable) (2.14) implies

$$E\overline{M}_m \leq a + \sum_{k=1}^{m} \int_a^\infty R_k(x)dx \ . \tag{2.14}$$

Minimizing the right-hand side (RHS) of (2.14) with respect to $a$ yields (2.7) and (2.8) with the optimal $a_m$ given by (2.7).

For (ii) we use the fact

$$\underline{M}_m \geq b + \sum_{k=1}^{m} [Y_k - b]^-$$

where $a^- = \min\{a, 0\}$ and follow the above reasoning. (Alternatively, simply note that $\max\{Y_1,\ Y_2\ ,...,\ Y_m\} = \min\{-Y_1,\ -Y_2\ ,...,\ -Y_m\}$ and apply (i) ).

Finally for part (iii) we apply Boole's inequality to get

$$\Pr\{M_m > x\} = \Pr\{Y_1 > x, \ \text{or}\ ,....,\ \text{or}\ ,Y_m > x\} \leq \sum_{i=1}^{m} R_i(x) \ . \tag{2.15}$$

Let $x = (1+\varepsilon)a_m$ where $\varepsilon > 0$ and $a_m$ be defined by (2.7). Then, (2.12) with $c = 1+\varepsilon$ implies $R((1+\varepsilon)a_m) = o(1)R(a_m)$. This and (2.15) finally lead to

$$\Pr\{M_m \geq (1+\varepsilon)a_m\} = o(1)\sum_{i=1}^{m} R(a_m) = o(1)$$

where the last equality comes from the definition (2.7). ∎

We now use Lemma 1 to estimate the height $EH_n = \max_{1\leq i<j\leq n}\{C_{ij}\}$. Note that there are $m = n(n-1)/2 \sim n^2/2$ random variables, namely, $n-1$ variables $C_1$, $n-2$ variables $C_2$ ,..., and one variable $C_{n-1}$. Proposition (i) gives the probability $\Pr\{C_d \geq k\}$. To estimate $a_n$ for $EH_n$, we use (2.7) which is now written:

$$\sum_{d=1}^{n}(n-d)R_d(a_n) = 1 \tag{2.16}$$

which we must solve for large values of $n$. For further analysis we need a rough estimate of $a_n$. We show that $a_n = O(\log n)$. Indeed, first note that the formula (2.3) for the function $R_d(k)$ in the binary case can be estimated as

$$R_d(k) = \left(p^{\lfloor \frac{k}{d}\rfloor+2} + q^{\lfloor \frac{k}{d}\rfloor+2}\right)^r \left(p^{\lfloor \frac{k}{d}\rfloor+1} + q^{\lfloor \frac{k}{d}\rfloor+1}\right)^{d-r} \leq \frac{n^2}{2}p_{\max}^k \tag{2.17}$$

where $p_{\max} = \max\{p,q\}$. Hence, $a_n \leq \frac{2}{\log p_{\max}^{-1}}\log n$ as needed.

Now, we are ready to prove (2.4) in Proposition (ii). The clue to the proof is to observe that for $d > k$ suffixes, say $S_i$ and $S_{i+d}$, of size $k$ are independent. This implies the following simplification in the formula on $R_d(k)$

$$k < d \qquad \Rightarrow \qquad R_d(k) = \Pr\{C_d \geq k\} = P^k \ , \tag{2.18}$$

where, to recall, $P = p^2 + q^2$. Using this fact, we split the RHS of (2.16) as follows

$$\begin{aligned}\sum_{d=1}^{n}(n-d)R_d(a_n) &= \sum_{d=1}^{a_n-1}(n-d)R_d(a_n) + \sum_{d=a_n}^{n}(n-d)R_d(a_n) \\ &= \sum_{d=1}^{a_n-1}(n-d)R_d(a_n) + P^{a_n}\sum_{d=a_n}^{n}(n-d) \ .\end{aligned}$$

Using the fact that $a_n = O(\log n)$ we easily show that $a_n$ must solve asymptotically the following equation (cf. (2.16))

$$\frac{n^2}{2}P^{a_n} + \sum_{d=1}^{a_n-1}(n-d)R_d(a_n) = 1 \ . \tag{2.19}$$

To prove $a_n \sim -2\log_P n$ it suffices to show that the second term in (2.19) tends to zero for such a choice of $a_n$.

Consider the second term in (2.19) for $a_n \sim -2\log_P n$. We prove that there exists such $\delta > 0$ that

$$\sum_{d=1}^{a_n-1}(n-d)R_d(a_n) = O(\frac{\log n}{n^\delta}) \ . \tag{2.20}$$

Indeed, note that (2.3) implies

$$\sum_{d=1}^{a_n-1}(n-d)R_d(a_n) \le \sum_{d=1}^{a_n-1}(n-d)(p^{a_n+1}+q^{a_n+1}) \le O(n\log n)p_{\max}^{a_n} \ . \tag{2.21}$$

But, with $a_n \sim -2\log_P n$ we have $p_{\max} \sim \exp(\frac{-2}{\log_{p_{\max}} P} n)$, hence (2.21) implies (2.20) with $\delta = \frac{-2}{\log_{p_{\max}} P} - 1 > 0$.

In summary, by Lemma 1 we have proved that

$$EH_n \le a_n + \sum_{j=a_n}^{\infty}\sum_{d=1}^{n}(n-d)R_d(j) \tag{2.22}$$

with $a_n \sim -2\log_P n$. To establish (2.4a) in Proposition (ii) we only need to show that the second term of the RHS of (2.2) is bounded. To prove this note that $R_d(a_n+k) \le cR_d(a_n)p_{\max}^k$ where $c > 0$ is a constant. Then,

$$\begin{aligned}\sum_{j=a_n}^{\infty}\sum_{d=1}^{n}(n-d)R_d(j) &= \sum_{k=0}^{\infty}\sum_{d=1}^{n}(n-d)R_d(a_n+k) \\ &\le c\sum_{d=1}^{n}(n-d)R_d(a_n)\sum_{k=0}^{\infty}p_{\max}^k = c\sum_{k=0}^{\infty}p_{\max}^k = C\end{aligned}$$

as need for Proposition (ii).

Finally, the convergence in probability (2.5) of Proposition (ii) follows immediately from Lemma 1 (iii) after noting that $R_d(k)$ satisfies (2.11). This completes the proof of our Proposition.

## 3. APPLICATIONS

Many combinatorial and algorithmic problems on words can be re-phrased in terms of self-alignments. Here, we focus on problems that involve a peculiar digital search index associated with a string. This index represents, in particular, a compendium of the self-alignments of that string, and has been proposed in various versions (see [AA] for detailed references). Here, we shall adopt the version known as *suffix tree*, introduced in [MC]. In this section, we analyze the impact of our probabilistic analysis on questions that revolve around the structure, construction and more or less sophisticated utilizations of suffix trees.

Given a string $X$ of length $n$ on the alphabet $\Sigma$, and a symbol \$ not in $\Sigma$, the *suffix tree* $T_X$ associated with $X$ is the digital search tree that collects the first $n$ suffixes of $X\$$. In the *noncompact* representation of $T_X$, each arc is labeled with a symbol of $\Sigma$, except for terminal arcs, that are labeled with a substring of $X\$$. The space needed can be $\Theta(n^2)$ in the worst case [AH]. In the *compact* representation of $T_X$, chains of unary nodes are collapsed into single arcs, and every arc of $T_X$ is labeled with a substring of $X\$$. A pair of pointers to a common copy of $X$ can be used for each arc label, whence the overall space taken by this version of $T_X$ is $O(n)$. In both representations, suffix $S_i$ of $X\$$ $(i = 1, 2, ..., n)$ is described by the concatenation of the labels on the unique path of $T_X$ that leads from the root to leaf $i$. Similarly, any vertex $\alpha$ of $T_X$ distinct from the root describes a subword $W(\alpha)$ of $X$ in a natural way: vertex $\alpha$ is called the *proper locus* of $W(\alpha)$. In the compact $T_X$, the *locus* 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 $W$.

Our probabilistic analysis yields bounds on the *structure* of the noncompact version of $T_X$. Such bounds hold *a fortiori* for the compact version. The bounds stem from the fact that (cf. also [SZ2]) the expected height $H(T_X)$, the average depth $D(T_X)$ and the shortest path $h(T_X)$ are simply related, respectively, the $n$-th height, depth and shallowness of any unbounded word $Z$ having $X$ as a prefix. In terms of such $Z$, we have $H(T_X) \leq H_n + 1$, $D(T_X) \leq D_n + 1$, and $h(T_X) \leq h_n + 1$. Thus, for a random suffix tree the expected height, the average depth and the shortest path are bounded by $2\log_b n + o(\log n)$, where $b^{-1} = P = \sum_{i=1}^{V} p_i^2$ is the probability of a match. While the first two bounds are tight for large $n$, the third one is never so, since the shortest path in any suffix tree has length 1 by construction. (Refer to [JS] for more precise estimates of the average depth.) We derive the that the expected external path length is bounded by $n\log_b n + o(n\log n)$. Finally, it is not difficult, using our analysis, to prove that the average number of nodes is $O(n)$ (cf. [JS]).

We now turn the *construction* and *uses* of $T_X$. erom now on, we assume a bounded $\Sigma$. Removing this assumption yields an extra multiplicative factor of $\log|\Sigma|$ in all time bounds that we mention. The *direct* or *brute force* construction of $T_X$ consists of starting with the empty tree $T_0$ and then inserting the suffixes of $X$ one by one into consecutive updates of the tree, as follows

$$\textbf{for } i := 1 \textbf{ to } n \textbf{ do } T_i \leftarrow \text{insert } (T_{i-1},\ S_i)\ .$$

Straightforward implementation of *insert*$(T_{i-1}, S_i)$ leads to a $\Theta(n^2)$ worst-case time construction of $T_X$ (cf. [MC] and [AH, Chapt. 9]). Such an implementation consists of starting at the root of $T_{i-1}$ and then following the edges whose labels describe the longest prefix $head_i$ of $S_i$ such that $head_i$ has a locus in the tree. This process can be carried out without ever forming chains of unary nodes in the tree. However, the work charged by $insert(\ T_{i-1}, S_i)$ is still proportional to the length of $head_i$, since locating the locus of $head_i$ requires tracking down $head_i$ in $T_{i-1}$ symbol-by-symbol, irrespective of which representation of $T_X$ (i.e., noncompact or compact) is being built.

This observation enables to use our Proposition of Section 2 to derive the expected time required for the direct construction of $T_X$. In fact, $|head_i| = \max_{j<i} C_{ij}$, whence by Lemma 1 the average length of $head_i$ is $O(\log i)$. Thus, building $T_X$ by brute force requires $O(n \log n)$ time on average ( i.e., the expected value of the external path length; see above).

Clever constructions such as in [MC] avoid the necessity of tracking down each suffix starting at the root. The crucial fact used is that if $head_i = aW$ $(i = 1, 2, ...n)$ with $a \in \Sigma$, then $W$ is a prefix of $head_{i+1}$. However, the exploitation of this fact requires that some rather bulky auxiliary structures be introduced and managed during the construction of $T_X$. Even when the current update of the tree and its auxiliary attachments can be kept in the main memory throughout the construction, the management of auxiliary structures renders the constant hidden in the time complexity significantly larger than that involved in the direct construction. When, as is often the case, tree and auxiliary structures become rapidly too large to fit in the main memory, the traffic to and from secondary storage may beset the advantages of the asymptotically more efficient algorithm.

We now analyze the implications of our analysis on some structural and algorithmic problems on words whose solutions rely on $T_X$. A feature common to most of these problems is that their solutions require some postprocessing of $T_X$ if the tree is built by the linear time algorithm, while such solutions could be easily embodied in the direct construction. We divide the applications of $T_X$ into two classes. In the first class, that we call of *direct* applications, we place problems that have linear time solutions provided that $T_X$ is built in linear time. For these applications, our probabilistic analysis of the brute force construction of $T_X$ leads to a time performance that can be practically quite close to that the more sophisticated methods, but never matches that performance in the asymptotic sense. In the second class, that we call of *advanced* applications, the asymptotic expected time performance associated with the brute force approach matches and can be even better than that achieved by more elaborate approaches.

### 3.1 Direct Applications

The main direct applications of $T_X$ are in (i) the construction of inverted files for on-line pattern matching and (ii) some important universal data compression schemes. We analyze

(i) first.

By treating $T_X$ as the state transition diagram of a finite automaton it is possible to decide whether or not any given *pattern* $W$ occurs in $X$, in $O(|W|)$ time. The overall cost of building $T_X$ (preprocessing) and performing many queries on it can be thus advantageous over conventional linear time pattern matching. Irrespective of the type of construction used for $T_X$, one can always maintain that each vertex of $T_X$ bears the label of the smallest leaf in its subtree. Then, it is possible to find in $O(|W|)$ steps for arbitrary $W$ what is the first occurrence of $W$ in $X$ (in particular, this answers whether $W$ is a prefix of $X$). To find the last occurrence of $W$ in $X$ in $O(|W|)$ time for any $W$ requires a walk through $T_X$, after its linear time construction, but $T_X$ can be easily prepared for such queries during the brute force construction. Irrespective of the type of construction, the problem of finding all occurrences of $W$ can be solved in time proportional to $|W|$ plus the total number of occurrences (either visit the subtree of $T_X$ rooted at the locus of $W$ or preprocess $T_X$ once for all by attaching to each node the list of the leaves in the subtree rooted at that node). Along the same lines, one can weight each node of $T_X$ with the number of leaves in the subtree rooted at that node. This weighted version serves then as a statistical index for $X$, in the sense that, for any $W$, we can find the frequency of $W$ in $X$ in $O(|W|)$ time. This weighting cannot be embedded in the linear time construction of $T_X$, while it is trivially embedded in the brute force construction: Attach a counter to each node; then, each time a node is traversed during *insert*, increment its counter by 1; if *insert* culminates in the creation of a new node $\beta$ on the arc $(\text{FATHER}(\alpha), \alpha)$, initialize the counter of $\beta$ to 1 + counter of $\alpha$.

There are other straightforward uses of $T_X$, such as finding the longest repeated substring in $X$, finding the position identifier of a given position, etc., for which we refer to [AA, AH], and for which the average time complexity is associated with the height or depth of $X$ discussed in our Proposition of Section 2.

We turn now to (ii). The suffix tree $T_X$ is the natural habitat for a class of sequential data compression techniques based on textual substitution. This class embodies the few optimization problems in the realm of textual substitution that can be solved in polynomial (actually linear) time. Moreover, the techniques in this class also feature asymptotic optimality in the information theoretic sense [LZ, ZL].

The idea is that of interweaving the construction of a (possibly partial) suffix tree with a *parse* of the textstring into *phrases*, where each phrase is susceptible of a compact encoding. More precisely, suppose that we have compressed the prefix of $X$ up to position $i$, and let $P_{i-1}$ be the encoded version of this prefix. By definition, the prefix $head_i$ of $S_i$ occurred already in $X$. Thus, $head_i$ can be encoded simply by a pair of pointers, say, to the starting and ending position of this previous occurrence in $X$. Appending this pair to $P_{i-1}$ yields $P_i$, and the process continues. One byproduct of our analysis confirms the intuition that a "very random" string is not compressible. For such a sequence, the expected length of each phrase is $2 \cdot \log n$, i.e., exactly the length in bits of the pair of pointers! (In practice, efficient representations of integers in an unbounded range, like those in, e.g., [AF, ER, RPE], may be used to reduce the length of such pointer-encodings.)

### 3.2 Advanced applications

We analyze here (i) the problems of testing the square-freedom of a string $X$ or finding all squares in it, and (ii) the related problem of building indexes for the statistics without overlap of all substrings of a string $X$.

We examine (i) first. A *square* of $X$ is a word in the form $WW$, where $W$ is a *primitive* word, i.e., a word that cannot be expressed in any way as $V^k$ with $k > 1$ and $V$ nonempty. Square free words, i.e., words that do not contain any square subwords have attracted attention since the early works by A. Thue in 1912 [TH]. A copious literature, impossible to report here, has been devoted to the subject ever since. Clearly, an indefinitely long square free word cannot be built on a binary alphabet, but Thue found that such a string can be constructed on an alphabet with at least three symbols. Before addressing some of the algorithmic issues on squares, it seems of interest to see that our analysis accommodates this discontinuity.

Let $P_{sf}$ be the probability of not having any square subword that starts at some *given position* of an unbounded word $X$ on a $V$-ary alphabet $\Sigma$. If the position chosen is, say, position 1, then it is easy to see that $P_{sf}$ can be expressed in terms of the random variables $\{C_d\}_{d=1}^{\infty}$ defined in Section 2 as

$$P_{sf} = \Pr\{C_1 \le 0, C_2 \le 1 \ , ..., \ C_d \le d-1 \ , ... \ \} \ . \tag{3.1}$$

The evaluation of this *joint* probability is extremely difficult, but we can obtain a simple estimate of it. We appeal to the following lemma.

**Lemma 2**. *For any sequence of random variables $X_1 \ , X_2 \ , ..., \ X_n$ the following holds*

$$1 - \sum_{k=1}^{n} \Pr\{X_k > x_k\} \le \Pr\{X_1 \le x_1, X_2 \le x_2 \ , ..., \ X_n \le x_n\} \le \Pr\{X_1 \le x_1\} \ . \tag{3.2}$$

**Proof**. The RHS of (3.2) is trivial. For the LHS we obtain ( cf. [FE] )

$$\Pr\{X_1 \le x_1 , ..., X_n \le x_n\} = 1 - \Pr\{X_1 > x_1 \text{ or } X_2 > x_2 \text{ ... or } X_n > x_n\} \ge 1 - \sum_{k=1}^{n} \Pr\{X_k > x_k\} ,$$

as needed for the lemma. ∎

By Lemma 2, we can estimate our joint probability $P_{sf}$ in (3.1) by computing $\Pr\{C_1 = 0\}$ and $\Pr\{C_l > l - 1\}$ for all $l > 1$. But by our Proposition (i) formula (2.2), we immediately find

$$\Pr\{C_1 = 0\} = 1 - P$$

$$\Pr\{C_d > d - 1\} = P^d$$

where $P = \sum_{i=1}^{V} p_i^2$. Therefore,

$$\frac{1 - 2P}{1 - P} \le P_{sf} \le 1 - P \ . \tag{3.3}$$

In the case of uniform distribution, we have $P = 1/V$, and

$$\frac{V - 2}{V - 1} \le P_{sf} \le \frac{V - 1}{V} \ . \tag{3.4}$$

Note that for binary alphabet $V = 2$, $0 \le P_{sf} < 0.5$. But we know that, in this case, $P_{sf} = 0$, so the lower bound is achievable. On the other hand, for any $V > 2$ we have $P_{sf} \ge 0.5$, so with positive probability we can construct square free words over an $V$-ary alphabet with $V > 2$. Note also, that for larger $V$, the bounds in (3.4) are tight. For example, for $V = 5$, $0.75 \le P_{sf} \le 0.8$. This suggests that, for *most* random strings, a square is an unlikely event to occur at *any fixed position.*

We now return to the algorithmic problems. By marking all nodes leading to $S_1$ it is possible to spot all square prefixes of $X$ as a byproduct of the construction of $T_X$. The same straightforward strategy can be used for square suffixes. On the other hand, efficient algorithms for testing square-freedom or detecting all squares in $X$ require quite elaborate constructions [ML,CR,AP]. The number of distinct occurrences of squares in a word can be $\Theta(n \log n)$, which sets a lower bound for all algorithms that find all squares [CR]. For instance, infinitely many *Fibonacci words*, defined by:

$$W_0 = b \ \ ; \ \ W_1 = a$$

$$W_{m+1} = W_m W_{m-1} \qquad \text{for } m > 1$$

have $\Theta(n \log n)$ distinct *occurrences* of square subwords. Interestingly, the same bound holds for the number of distinct square subwords in $W_m$. The algorithms [ML, CR, AP] find all squares in $X$ in $O(n \log n)$, hence, optimal time. The construction of [AP] uses suffix trees in conjunction with the following criterion: $X$ contains a square occurrence at position $i$ iff there is a primitive word $W$ and a vertex $\alpha$ in $T_X$ such that $i$ and $j = i + |W|$ are consecutive leaves in the subtree of $T_X$ rooted at $\alpha$ and, moreover, $|W(\alpha)| \geq (i - j)$.

It is an easy exercise to embed such a criterion in the brute force construction of $T_X$. If, on the other hand, linear time construction is used, then the following postprocessing is necessary. Starting from the leaves of $T_X$, we visit the tree bottom-up. For each interior vertex visited we construct the sorted list of the labels of its leaves. The sorted list of any such vertex is obtained by merging the sorted lists of its offspring vertices. The strategy runs in $O(n \log n)$ time if $T_X$ is nearly balanced or completely unbalanced. Optimal handling of intermediate cases involves a rather complicated construction that makes use of an *ad hoc* data structure suited to the efficient repeated merging of integers in a known range [AP]. On the other hand, the above brute force implementation of the same criterion leads, by our probabilistic analysis, to an optimal (i.e., $O(n \log n)$) expected performance.

We devote the remainder of this section to problem (ii). The (primitive rooted) squares possibly existing in $X$ affect the amount of storage needed to allocate the statistics without overlap of all substrings of $X$. By this, we mean the construction of an index similar to $T_X$, but such that, given any word $W$, we can find in $O(|W|)$ time the maximum number $k$ of distinct occurrences of $W$ such that it is possible to write $X = W_1 W W_2 W W_3 ... W W_{k+1}$ with $W_d$ possibly empty $(d = 1, 2, ..., k+1)$.

The construction of such an index requires inserting a number of auxiliary unary nodes in $T_X$. The role of such nodes in the augmented tree is to serve as proper loci for subwords whose loci in the original tree cannot report the number of their nonoverlapping occurrences in $X$. We refer to [AA, AP1, AP2] for details. The relation between the auxiliary nodes of $T_X$ and the squares in $X$ is as follows [AP1]. If $\alpha$ is an auxiliary node of the augmented $T_X$, then there are subwords $U$ and $V$ in $X$ and an integer $k \geq 1$ such that $W(\alpha) = U = V^k$ and there is a substring $W$ in $X$ such that $W = V^m V'$ with $V'$ a prefix of $V$ and $m \geq 2k$. An $O(n \log n)$ upper bound on the number of auxiliary nodes needed in $T_X$ can be readily set, based on the above fact and on the upper bound on the number of positioned squares in a word. However, it is an interesting open question whether there are words whose minimal augmented suffix trees do in fact attain that bound. Auxiliary nodes can be inserted and weighted through a fairly complex, $O(n \log^2 n)$ postprocessing of $T_X$, once the tree has been built in linear time [AP1, AP2]. On the other hand, these manipulations can be carried out along with the brute force construction of $T_X$, with no substantial penalty. It follows from our analysis that, from the probabilistic view point the brute force construction of an augmented suffix tree can be expected to be asymptotically faster by a factor of $\log n$ in comparison to the more advanced constructions.

## 4. CONCLUDING REMARKS

It seems interesting to compare the basic parameters of suffix trees and radix search trees (in short tries)[AH, KN]. In tries $n$ *independent* keys $X_1, X_2, ..., X_n$ are stored, where each key is a (possibly infinite) sequence of symbols over a $V$-ary alphabet. Note that in suffix trees the keys $S_1, S_2, ..., S_n$ are *dependent* while in tries it is assumed that the keys are statistically independent. One may ask how close a random shape of a suffix tree resembles the shape of independent tries.

In order to compare suffix trees and independent tries we need some results for tries. Fortunately, in recent years independent tries have been studied very extensively (cf. [FL, KN, PI, SZ1, SZ2]). In particular, it is proved that the average depth for tries is $ED_n = \frac{1}{E}\log n + O(1)$ where $E$ is the entropy of the alphabet, that is, $E = -\sum_{i=1}^{V} p_i \log p_i$. For the average height the following result is known (cf. [FL, PI, SZ2]): $EH_n = \frac{2}{\log P^{-1}}\log n + O(1)$ where $P = \sum_{i=1}^{V} p_i^2$. Moreover, for independent keys it is proved that the variance of the depth $varD_n$ is either $O(1)$ for symmetric alphabet (e.g. $varD_n = 3.507...$ for $V = 2$; cf. [SZ1]) or $D_n = \frac{E_2 - E^2}{E^3}\log n + O(1)$, for the asymmetric tries, where $E_2 = \sum_{i=1}^{V} p_i \log^2 p_i$. This implies that asymmetric tries are of an order of magnitude less balanced that the symmetric ones. We note, however, that our Proposition and simulation results (see Table 1 in Section 2) suggest that asymptotically suffix trees and independent tries do *not* differ too much: they have the same order of growth up to the leading term in the asymptotics for the height and the depth. In fact, this circumstances were recently established in [DSR] and [JS], respectively. An open question is still how to evaluate the variance and the limiting distribution of the height for suffix trees.

Another issue of some interest is how much the compact version of suffix trees is better than the noncompact one. We can answer that question indirectly comparing regular tries (independent keys) with the compact version of the trie known as PATRICIA trie (also with independent keys). In [SZ3] it is proved that the average depth for PATRICIA is $ED_n = \frac{1}{E}\log n + O(1)$, hence the difference between the depths of regular and PATRICIA tries $O(1)$. The variance of the depth is either $O(1)$ for symmetric case or $O(\log n)$ for asymmetric PATRICIA, exactly as in the regular tries (see above). For example, for binary tries and PATRICIA tries the variances are either 3.507.. or 1.00.. respectively [SZ3]. It is also known that for symmetric PATRICIA the average height is asymptotically equal to $\log_V n$ rather than $2 \cdot \log_V n$ as for regular tries [PI].

## REFERENCES


[AA] A. Apostolico, The Myriad Virtues of Suffix Trees, *Combinatorial Algorithms on Words,* pp. 85–96, Springer-Verlag, ASI F12 (1985).

[AF] A. Apostolico and A.S. Fraenkel, Robust Transmission of Unbounded Strings Using Fibonacci Representations, *IEEE Transactions on Information Theory,* IT-23, 238–245 (1987).

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

[AP] A. Apostolico and F.P. Preparata, Optimal Off-line Detection of Repetitions in String, *Theoretical Computer Science,* 22, 287–315 (1983).

[AP1] A. Apostolico, P.F., Preparata, Structural Properties of The String Statistics Problem, *Journal of Computer and Systems Science*, 31, 2, 394-411 (1985).

[AP2] A. Apostolico, F.P., Preparata, Data Structures and Algorithms for the String Statistics Problem, Purdue CS D-TR-547 (1985).

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

[CR1] M., Crochemore, Recherche Lineaire d'un Carre dans un Mot, *C.R. Acad. Sc. Paris*, t.296, Series I, 781-784 (1983).

[DSR] L. Devroye, W. Szpankowski and B. Rais, A note of the height of suffix trees, Purdue University, CSD TR-905 (1989); submitted also to a journal.

[ER] S. Even and M. Rodeh, Economical Encoding of Commas between Strings, *Comm. ACM* 21, 315-317 (1978).

[FE] W. Feller, *An Introduction to Probability Theory and its Applications,* John Wiley & Sons (1968).

[FL] P. Flajolet, On the Performance Evaluation of Extendible Hashing and Trie Searching, *Acta Informatica,* 20, 345–369 (1983).

[GO] L. Guibas and A. W. Odlyzko, String Overlaps, Pattern Matching, and Nontransitive Games, *Journal of Combinatorial Theory,* Series A, 30, 183-208 (1981).

[JS] P. Jacquet and W. Szpankowski, Autocorrelation on Words and Its Applications. Analysis of Suffix Tree by String-Ruler Approach, INRIA TR-1106 (1989); also submitted to a journal.

[KN] D. Knuth, *The Art of Computer Programming, Sorting and Searching,* Addison-Wesley (1973).

[LO] M. Lothaire, *Combinatorics on Words,* Addison-Wesley (1982).

[LR1] T. Lai and H. Robbins, Maximally Dependent Random Variables, *Proc. Nat. Acad. Sci. USA,* 73, 286–288 (1976).

[LR2] T. Lai and H. Robbins, A Class of Dependent Random Variables and Their Maxima, *Z. Wahrscheinlichkeitscheorie,* 42, 89–111 (1978).

[LZ] A. Lempel and J. Ziv, On the Complexity of Finite Sequences, *IEEE Information Theory* 22, 1, 75-81 (1976).

[MI] D.S. Mitrinović, *Analytic Inequalities,* Springer-Verlag (1970).

[MC] E.M. McCreight, A Space Economical Suffix Tree Construction Algorithm, *JACM*, 23, 262–272 (1976).

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

[ML1] M. G. Main and R. J. Lorentz, Linear Time Recognition of Square-Free Strings, *Combinatorial Algorithms on Words*, (A. Apostolico and Z. Galil, eds.) Springer-Verlag (1984).

[PI] B. Pittel, Asymptotic growth of a class of random trees, *The Annals of Probability,* 18, 414 - 427 (1985).

[RPE] M. Rodeh, V. Pratt and S. Even, Linear Algorithm for Data Compression via String Matching, *Journal of the ACM*, 28, 16-24 (1981).

[SZ1] W. Szpankowski, Some Results on $V$-ary Asymmetric Tries, *Journal of Algorithms,* 9, 224-244 (1988).

[SZ2] W. Szpankowski, On the height of Digital Trees and related Topics, *Algorithmica*, 5 (1991).

[SZ3] W. Szpankowski, Patricia tries again revisited, *Journal of the ACM,* 37, 691–711 (1990).

[TH] A. Thue, Uber die gegenseitige Lage gleicher Teibe gewisser zeichenreichen, *Skr. Vid. Kristiana 1, Math Naturv.* Klasse 1, 7–67 (1912).

[WE] P. Weiner, Linear Pattern Matching Algorithms, *Proc. of the 14-th Annual Symposium on Switching and Automata Theory,* 1–11 (1973).

[ZL] J. Ziv and A. Lempel, A Universal Algorithm for Sequential Data Compression, *IEEE Information Theory*, 23, 3, 337-343 (1977).