1987

Self-Alignments in Words and Their Applications

Alberto Apostolico

Wojciech Szpankowski
Purdue University, spa@cs.purdue.edu

Report Number:
87-732

SELF-ALIGNMENTS IN WORDS AND THEIR APPLICATIONS

Alberto Apostolico
Wojciech Szpankowski

CSD-TR-732
December 1987

SELF-ALIGNMENTS IN WORDS AND THEIR APPLICATIONS

*Alberto Apostolico and Wojciech Szpankowski**

Department of Computer Sciences
Purdue University
West Lafayette, IN 47907

Abstract

This paper deals with the probabilistic analysis of some quantitative measures associated with periodicities in words, notably, the various values attained by the lengths of the longest common prefix of pairs of suffixes of a given word. Such values, that are called here *self-alignments*, play a crucial role in several algorithmic constructions, such as building the suffix tree or inverted file of a word, detecting squares and other repetitions in a word, computing substring statistics, etc. The probabilistic analysis of self-alignments is then used to study the expected time complexities of straightforward algorithmic solutions to these problems, and to compare such performances with those attained by more complex constructions.

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

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 algorithmic constructions have been set up to date both to detect and exploit the presence of repeated subpatterns and other kinds of more or less unavoidable regularities in words. 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 speed with which the following basic question is, once or repeatedly, answered: 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 optimal solutions. For others, efficient solutions are available that may nevertheless be susceptible of further improvements. For all these problems, however,

* Supported in part by NSF under grant NCR-8702115

algorithmic design was so far mostly finalized to the optimization of the asymptotic worst-case behavior. As is often the case, the constructions resulting from this endeavor are generally quite elegant, but also quite involved. In general, this inflates the constants hidden in the corresponding figures of asymptotic performance. By contrast, straightforward constructions exist that appear conceptually rather naive, but do not present, even in the asymptotic sense, an unbearable computational overhead with respect to the more elaborate solutions. Since the worst cases for these problems are often represented by rather unrealistic, even pathological inputs, it seems natural to inquire about the expected performance of their naive algorithmic solutions, and compare such performances with those of more clever methods. The results of this paper suggest that, under reasonable probabilistic assumptions, the straightforward algorithms for the problems on words that are considered here have an expected asymptotic time complexity that is for some problems only slightly worse, and for some other problems equal or even better than the time complexity of the corresponding clever solutions.

This paper is organized as follows. In Section 2, we introduce some measures for what can be loosely defined as correlations among subwords of a given word. We find it convenient to assume such a word unbounded, but the upper bounds that we derive based on this assumption will hold *a fortiori* for strings of finite length. In particular, we derive the distribution function of the longest common prefix of two suffixes of a given word. Using this we prove that the average values of the largest and the average longest common prefix of all suffixes (the so called height and depth respectively) are $O(\log n)$. In Section 3, we apply our probabilistic results to the average case analysis of the straightforward versions of some important algorithms on words. We summarize the main results of that section, referring to the case of a binary input string emitted by a symmetric source. For such a string, we find that building the suffix tree or inverted file associated with a word, which takes linear time by clever methods [MC], takes $O(n \log n)$ time by the naive method; detecting all squares in a word, which takes optimal $O(n \log n)$ time by clever methods [AP, CR, ML], takes $O(n \log n)$ expected time by the direct method; computing the full statistics without overlap of all substrings of a word, which takes $O(n \log^2 n)$ time by clever methods [AP1, AP2], takes $O(n \log n)$ expected time by the direct method, etc. The same asymptotic bounds hold in the case of nonuniform distributions, although the constants involved grow with the highest probability associated with a source symbol. Section 4 concludes our discussion by relating the present results to those obtained by previous studies on general tries [SZ1, SZ2, SZ3].

2. AUTOCORRELATION PARAMETERS IN WORDS

In this section, we introduce some basic definitions and present a thorough 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 Σ of cardinality V, and let $S_i = x_i x_{i+1} \ldots$ be the i-th *suffix* of X, i=1,2,.... 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 three expressions define, in succession, the n-th *height* H_n of X, the n-th *shallowness* h_n of X, and the n-th *depth* D_n of X.

$$H_n = \max_{1 \le i < j \le n} \{C_{ij}\}, \tag{2.1a}$$

$$h_n = \min_{1 \le i \le n} \left\{ \max_{1 \le j \le n,\, j \ne i} \{C_{ij}\} \right\}, \tag{2.1b}$$

$$D_n = \sum_{i=1}^{n} \frac{\max_{1 \le j \le n,\, j \ne i} \{C_{ij}\}}{n}. \tag{2.1c}$$

Intuitively, H_n measures the length of the longest substring Z of X that starts at some position $j \le n$ of X and such that the occurrence of Z that starts at j can be fully recopied from some previous occurrence of Z in X. The depth D_n represents the average length of the string Z which can be recopied. The height H_n and its companion parameters express mutual 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 reminisce in various ways 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 $X = abbabaa\ldots$ and $n = 5$. Then $S_1 = X$, $S_2 = bbabaa\ldots$, $S_3 = babaa\ldots$, $S_4 = abaa\ldots$ and $S_5 = baa\ldots$. The corresponding self-alignment matrix $C = \{C_{ij}\}$, i=1,2,...5;,j=1,2,...5 is as follows:

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

From C and the expressions (2.1), we obtain $H_n = 2$, $h_n = 1$, $D_n = 9/5$.

□

In some applications, another quantity based on the self-alignment matrix arises, namely:

$$\chi_n = \sum_{i=1}^{n} \frac{\min\limits_{1 \le j \le n} \{C_{ij} \mid C_{ij} > 0\}}{n} \qquad (2.1d)$$

Hence, χ_n measures the row-wise average of the row-by-row minima attained by all and only the positive commons.

We deal here with the probabilistic analysis of the above quantities under the *Bernoulli* assumptions: *the symbols of X are drawn independently from Σ, and the i-th symbol of Σ occurs in X with probability p_i, $i = 1, 2, \ldots, V$, $\sum_{i=1}^{V} p_i = 1$.* We first compute the distributions of all random variables C_{ij} $(i = 1, 2, \ldots, n, j = 1, 2, \ldots, n)$ in the Bernoulli model, and then we use such distributions to evaluate the average values EH_n, ED_n and Eh_n of the nth height, depth and shallowness of X, respectively. 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, and we denote this random variable by C_d. For example, $C_{1,2}, C_{2,3}, \ldots, C_{n-1,n}$ have the same distribution as C_1 (i.e., $d = 1$). Thus, it is appropriate to reason in terms of the random variables C_d, where $d = 1, 2, \ldots, n-1$. We remark, however, that, the random variables in a family such as $C_{1,d+1}, C_{2,d+2}, \ldots, C_{n-d,n}$ *are* dependent.

Our main results of this Section are summarized in the following proposition.

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

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

where $k = 0, 1, \ldots,$ and p_i is the probability of selecting the i-th symbol from the alphabet Σ. For the symmetric distribution $p_1 = p_2 = \cdots = p_V = 1/V$, expression (2.2a) simplifies to

$$Pr\{C_d = k\} = \left[\frac{1}{V} \right]^k \left[1 - \frac{1}{V} \right] \tag{2.2b}$$

(ii). The n-th average height EH_n satisfies

$$EH_n \le \frac{2}{\log p_{\max}^{-1}} \log n + c \tag{2.3a}$$

where log represents the natural logarithm, $p_{\max} = \max_{1 \le i \le V} p_i$, and c is a constant. In the symmetric case we have the stronger result

$$EH_n \sim 2 \log_V n \ , \tag{2.3b}$$

where ~ denotes "asymptotically equal to".

(iii). The following inequalities hold, respectively, for the average depth and shallowness

$$ED_n \le \frac{1}{\log p_{\max}^{-1}} \log n + c' \tag{2.4}$$

$$Eh_n \le \frac{1}{\log p_{\max}^{-1}} \log n + c'' \tag{2.5}$$

In the symmetric case,

$$ED_n \sim \log_V n \tag{2.4a}$$

$$Eh_n \sim \log_V n \tag{2.5b}$$

and, in addition,

$$E\chi_n \sim \log_V (1 - 1/n) \tag{2.5c}$$

□

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, ED_n and 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 $EH_n \sim (2/\log p_{\max}^{-1})\log n$. The constant $2/\log p_{\max}^{-1}$ becomes very large in strongly asymmetric cases, that is, for $p_{\max}$ close to one. On the other hand, the constant at $\log n$ in Eh_n is too large. Later, we indicate that this constant can be reduced to $1/\log p_{\min}^{-1}$, where $p_{\min} = \min_{1 \le i \le V} p_i$ and this seems to be asymptotically correct.

(iii) In practice, we are interested in finite strings in the form $X\$ = x_1 x_2 \cdots x_n \$$, where \$ is a symbol not in the alphabet Σ. Even though the suffixes of X are now finite, our results still hold in these cases, since one can consistently set $Pr\{C_{ij} = k\} = 0$ for $k > n-j$ in formulas (2.2).

•

2.2 Probability distribution function of C_d

We compute the probability distribution functions $Pr\{C_d = k\}$ for $d = 1, 2, \ldots, n-1$. To simplify notation, 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 by $p = p_1$ and $q = q_2 = 1 - p$, 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 $j-i=d$, and let $C_{ij}=k \ge 0$. Then, string Z which is a common prefix of S_i and S_j with $|Z| = k$ can be written as $Z = U^l U'$, where $|U| = d$, $|U'| = r < d$, and U' is a prefix of U, and U^l is the string resulting from the concatenation of l copies of U.

□

The probabilistic implications of the above fact are illustrated by the following example.

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

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

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

Figure 1.

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

CASE 1: $k \le d - 1 = 2$

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

$$Pr\{C_3 = 0\} = Pr\{x_1 \ne y_1\} = 1 - P$$

$$Pr\{C_3 = 1\} = Pr\{x_1 = y_1, x_2 \ne y_2\} = P(1 - P)$$

$$Pr\{C_3 = 2\} = Pr\{x_1 = y_1, x_2 = Y_2, x_3 \ne y_3\} = P^2(1 - P)$$

CASE 2. $3 = d \le k \le 2d - 1 = 5$

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

$$Pr\{C_3 = 3\} = Pr\{x_1 = y_1 \ne y_4,\ \ x_2 = y_2,\ \ x_3 = y_3\}$$

$$= Pr\{x_1 = y_1 \ne y_4\}\, Pr\{x_2 = y_2\}\, Pr\{x_3 = y_3\}$$

$$= [Pr\{x_1 = y_1 \ne y_4 | y_1 = 0\} \cdot Pr\{y_1 = 0\} + Pr\{x_1 = y_1 \ne y_4 | y_1 = 1\} \cdot Pr\{y_1 = 1\}] \cdot P^2$$

$$= (p^2 q + q^2 p)P^2$$

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

$$Pr\{C_3 = 4\} = Pr\{x_1 = y_1 = y_4, x_2 = y_2 \ne y_5, x_3 = y_3\} =$$

$$= Pr\{x_1 = y_1 = y_4\} \cdot Pr\{x_2 = y_2 \ne y_5\}\, Pr\{x_3 = y_3\}$$

$$= (p^3 + q^3)(p^2 q + q^2 p)P$$

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

$$Pr\{C_3 = 3l + 1\} = Pr\{x_1 = y_1 = \cdots = y_{3l+1}, x_2 = y_2 = \cdots \neq y_{3l+2}, x_3 = y_3 = \cdots = y_{3l}\}$$

$$= (p^{l+2} + q^{l+2})(p^{l+1} q + q^{l+1} p)(p^{l+1} + q^{l+1})$$

□

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 entirely predictable in its essence and 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 mutually independent d groups, and the value of r indicates in which group an inequality holds. Hence

$$Pr\{C_d = dl + r\} = Pr\{x_1 = y_1 = \cdots = y_{dl+1}, \ldots, x_{r+1} = y_{r+1} = \cdots \neq y_{dl+r+1}, \ldots,$$
$$x_d = \cdots = y_{dl}\} = (p^{l+2} + q^{l+2})^r (p^{l+1} q + q^{l+1} p)(p^{l+1} + q^{l+1})^{d-r-1} \tag{2.6a}$$

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

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

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

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

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).

2.3 The average height, depth and shallowness

In this subsection, we prove Proposition (ii) (formulas (2.3a)–(2.5)). To compute the average height, depth and shallowness, 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}, $i, j = 1, 2, \ldots, n$. 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 $Y_1, Y_2, \ldots, Y_m$ be a sequence of random variables with distribution function $F_1(y), F_2(y), \ldots, F_m(y)$, respectively. Let $R_i(y) = Pr\{Y_i \geq y\}$ be the *complement* function of the distribution function $F_i(y)$ (function R 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 a solution of

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

then

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

(ii) If b_m is a solution of

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

then

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

(iii) If $Y_1, Y_2, \ldots, Y_m$ are identically distributed with distribution (reliability) function $F(y)$ $(R(y))$, and, moreover,

$$\lim_{y \to \infty} \frac{1-F(cy)}{1-F(y)} = 0 \quad \text{for} \quad c > 1, \tag{2.11}$$

then

$$\lim_{m \to \infty} \frac{E\overline{M}_m}{a_m} = \lim_{m \to \infty} \frac{E\underline{M}_m}{b_m} = 1\ . \tag{2.12}$$

That is, $\overline{M}_m \sim a_m$ and $\underline{M}_m \sim b_m$ where a_m and b_m solve

$$m\ R(a_m) = 1 \quad \text{and} \quad m\ F(b_m) = 1, \tag{2.13}$$

respectively.

Proof. (i) Observe that, for any a (cf. [LR1]),

$$\overline{M}_m \leq a + \sum_{k=1}^{m} [Y_k - a]^+ \tag{2.14}$$

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

$$E\overline{M}_m \leq a + \sum_{k=1}^{m} \int_a^\infty R_k(x)dx \tag{2.15}$$

Minimizing the right-hand side (RHS) of (2.15) with respect to a yields (2.7) and (2.8) with the optimal a_m given by (2.7).

(ii) 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 $\max\{Y_1, Y_2, \ldots, Y_m\} = \min\{-Y_1, -Y_2, \ldots, -Y_m\}$ and apply (i)).

(iii) This part is much more difficult and is established in [LR2].

□

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 $C_2, \ldots,$ and one variable C_{n-1}. Proposition (i) gives the probability $Pr\{C_d = k\}$. To estimate a_n for EH_n, we use (2.7) which is now written:

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

with $R_d(k) = Pr\{C_d \geq k\}$. We first compute the reliability function $R_d(k)$ and then solve (2.16).

Theorem. With $k = dl + r$,

$$Pr\{C_d \geq k\} = R_d(k) = \left\{ \sum_{i=1}^{V} p_i^{l+2} \right\}^r \left\{ \sum_{i=1}^{V} p_i^{l+1} \right\}^{d-r} \tag{2.17}$$

Proof. The claim follows from an argument analogous to that used in establishing Proposition (i), once the condition that S_i and S_{i+d} must disagree on their k+1-st symbol is dropped. For example, in the binary case, we get (cf. (2.6a) and Figure 1):

$$Pr\{C_d \geq dl + r\} = Pr\{x_1 = y_1 = \cdots = y_{dl+1}, \ldots, x_{r+1} = y_{r+1} = \cdots = y_{dl}, \ldots, x_d = y_d = \cdots = y_{dl}\} =$$
$$(p^{l+2} + q^{l+2})^r (p^{l+1} + q^{l+1})^{d-r}. \tag{2.18}$$

□

Note that (2.18) can be re-written as

$$R_d(k) = (p^{f+2} + q^{f+2})^r (p^{f+1} + q^{f+1})^{d-r} \tag{2.19}$$

where f equals the integer part of the ratio k/d, denoted $\lfloor \frac{k}{d} \rfloor$. Then, a_n is a solution of (2.16) with $R_d(a_n)$ given by (2.17). Nevertheless, to obtain asymptotics for a_n, we need a simpler form of $R_d(k)$. From (2.19), one immediately gets

$$\underline{R}_d(k) \stackrel{\text{def}}{=} (p^{f+2} + q^{f+2})^d \le R_d(k) \le (p^{f+1} + q^{f+1})^d \stackrel{\text{def}}{=} \overline{R}_d(k) \tag{2.20}$$

Using (2.20) we estimate the solution a_n of (2.16) appealing to the following lemma.

Lemma 2. If $\underline{R}_d(k) \le R_d(k) \le \overline{R}_d(k)$ for all $k = 0, 1, \ldots,$ and $\underline{a}_n$ (resp, $\overline{a}_n$) is a solution of (2.16) with $R_d(k)$ replaced by $\underline{R}_d(k)$ (resp., $\overline{R}_d(k)$), then

$$\underline{a}_n \le a_n \le \overline{a}_n \tag{2.21}$$

Proof. This follows directly from the monotonicity of the reliability function $R_d(k)$.

$\square$

Our next step is to compute a_n from (2.16), after which, by (2.8)

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

We prove first that if a_n satisfies (2.16), then the second term in RHS of (2.22) is O(1). Note that, by (2.16) and (2.20),

$$\sum_{j=a_n}^{\infty} \sum_{d=1}^{n} (n-d) R_d(j) = \sum_{k=0}^{\infty} \sum_{d=1}^{n} (n-d) R_d(a_n+k) = O\left[\sum_{d=1}^{n} (n-d) R_d(a_n) \sum_{k=0}^{\infty} p_{\max}^{k} \right] = O(1) \tag{2.23}$$

To conclude our analysis, we need an estimate of the a_n which solves (2.16). Using the inequality (see [MI, Sect. 2.14])

$$[\, p\, p^{\lfloor \frac{a}{d} \rfloor} + q\, q^{\lfloor \frac{a}{d} \rfloor}\,]^d \le p^{a+1} + q^{a+1}$$

we find

$$\sum_{d=1}^{n} (n-d)R_d(a_n) \le m(p^{a_n+1} + q^{a_n+1}) \overset{\text{def}}{=} L(a_n) \qquad (2.24)$$

By Lemma 2 the solution a_n of (2.16) is upper bounded by a solution of $L(\overline{a}_n) = 1$, where $L(a_n)$ is the RHS of (2.24). The asymptotic value of $\overline{a}_n$ is given in the next lemma.

Lemma 3. Let f_m be a solution of

$$\alpha m(p^{f_m+1} + q^{f_m+1}) = 1 \qquad (2.25a)$$

Then for large m

$$f_m = \log_{p_{\max}}(\alpha m)^{-1} - 1 + O(1) \qquad (2.26b)$$

Proof. Let $m \to \infty$, and for simplicity assume $p_{\max} = p$. Then one finds

$$\lim_{m \to \infty} \alpha m \; p^{\log_p(\alpha m)^{-1} + O(1)} = 1$$

$$\lim_{m \to \infty} \alpha m \; q^{\log_p(\alpha m)^{-1} + O(1)} = \lim_{m \to \infty} (\alpha m)^{-\varepsilon} = 0$$

where $\log q / \log p = 1 + \varepsilon$, and $\varepsilon > 0$.

□

Lemma 1 and formula (2.23) of Lemma 3 complete the proof of formula (2.3a) in Proposition (ii). Formula (2.3b) follows immediately from the above discussion and part (iii) of Lemma 1.

To prove (2.4) of Proposition (iii), observe that $E \max_{1 \le j \le n \; i \ne j} C_{ij} = \dfrac{\log n}{\log p_{\max}^{-1}} + O(1)$ since only $n-1$ random variables are involved in the maximum operation. For Eh_n, we note that

$$E \min_{1 \le i \le n} \max_{1 \le j \le n,\, j \ne i} C_{ij} \le \min_{1 \le i \le n} E \{ \max_{1 \le j \le n,\, j \ne i} C_{ij} \}$$

and we can apply the previous analysis. In fact, in this case, it is not difficult to see that the minimum of $E \max_j C_{ij}$ is achieved when we follow the least likely path, hence, we can curb the constant at log n down to $1/\log p_{\min}^{-1}$.

Finally, to prove (2.5c) of Proposition (iii), we proceed as follows. Note that $\min_{1 \le i \le j \le n, j \ne i} \{C_{ij} \mid C_{ij} > 0\}$ is maximum in the symmetric case, so that we can restrict our analysis to that case. Then, $Pr\{C_d = k\} = p^k(1-p)$, $k = 0, 1, \ldots,$ and $p = 1/V$. To obtain the minimum of C_{ij} over all positive off-diagonal C_{ij}'s, set $C_{ij}' = \infty$ when $C_{ij} = 0$, and $C_{ij}' = C_{ij}$ otherwise. Then $Pr\{C_{ij}' = k\} = p^{k-1}(1-p)$ $k = 1, 2, \ldots,$, and the common distribution function for all i and j is $F(k) = 1 - p^k$. By Lemma 1 formula (2.13) b_n satisfies $n\, F(b_n) = 1$ which implies $b_n = \log_V(1 - 1/n)$. Appealing again to part (iii) of Lemma 1 completes the proof.

3. APPLICATIONS

Several important combinatorial and algorithmic problems on words can be posed and solved in terms of the self-alignments of a string. Such a strong degree of unification derives from the fact that the most efficient solutions known for these problems are supported by a peculiar notion of digital search index associated with a string, an index that represents, in particular, a compendium of the self-alignments of that string. Various incarnations have been proposed for such an index during the past decade, but they do not differ significantly in their substance and power. The interested reader is referred to [AA] for more information. Here we shall adopt the version known as *suffix tree*, originally introduced in [MC]. In this section, we analyze the impact of our probabilistic analysis on questions that revolve around the construction and the more or less sophisticated use of suffix trees and companion structures.

Given a string X of length $n-1$ on the alphabet Σ, and a symbol $\$$ not in Σ, the *suffix tree* T_X associated with X is the digital search tree that collects the n suffixes of $X\$$. In the *compact* version of T_X, chains of unary nodes are collapsed into single arcs. Each arc of T_X is labeled with a substring of $X\$$, in such a way that suffix S_i of $X\$$ 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 α of T_X distinct from the root describes a subword $W(\alpha)$ of X in a natural way: vertex α is called the *proper locus* of $W(\alpha)$. In general, the *locus* of W in T_X 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. A pair of pointers to a common copy of X is commonly used for each arc label, whence the overall space taken by T_X is $O(n)$.

In the following, we assume a bounded size for Σ. Removing this assumption yields an extra multiplicative factor of $\log|\Sigma|$ in all of the time bounds that we mention. The obvious approach to the construction of T_X is to start with the empty tree T_0 and insert the suffixes of X one by one into consecutive updates of the tree, as follows

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

Before we address some algorithmic issues of this suffix tree, we first discuss some complexity questions of the tree using our notion of self-alignments from Section 2.

It is easy to see that the straightforward implementation of *insert* may require $\Theta(n^2)$ overall time in the worst case (cf. also [AH, Chapt. 9). Such an implementation consists of starting at the root of the current version of the tree and then follow the edges whose labels describe the longest prefix $head_i$ of S_i such that $head_i$ has a locus in the tree. Although this process can be carried out without ever forming chains of unary nodes in the tree, the work charged by *insert*(T_{i-1}, S_i) is proportional to the length of the longest prefix of S_i that has a locus in T_{i-1}. In other words, this work is irrespective of whether the compact or noncompact version of T_X is being built. It should be obvious from our previous discussion that the noncompact version of T_X with each arc labelled by a symbol from the alphabet Σ upper bounds all other constructions of T_X. Moreover, for such a (digital) tree, using a slight modification of the arguments proposed in [SZ2], we can easily show that the height of the tree H_n^T, the depth of the tree D_n^T, and the shortest path in the tree h_n^T are simply related to the n-th height H_n of a word X, the n-th depth D_n, and the n-th shallowness h_n as defined in (2.1). Namely, $H_n^T = H_n + 1$, $D_n^T = D_n + 1$ and $h_n^T = h_n + 1$. This observation enables to use our Proposition of Section 2, in conjunction with Remark (iii), 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 also below). Along the same lines, our analysis implies that for a random suffix tree the average height is bounded by 2 $\log_a n + O(1)$, the average depth is $\log_a n + O(1)$, and the shortest path is $\log_b n + O(1)$, where $a = p_{\max}^{-1}$ and $b = p_{\min}^{-1}$. In particular, the average of the external path length is $n\ lgo_b n + O(n)$. Moreover, it is not difficult, using our analysis, to prove that the average number of nodes is $O(n)$.

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 render 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 risks to beset the advantages of having produced an asymptotically more efficient

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 that 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. 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. For example, 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 is a confirmation of the intuitively obvious fact 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!

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 on the form WW, where W is a *primitive* word, i.e., a word that cannot be expressed in any way as V^k with $k>1$. 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 Σ. 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

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

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 4. For any sequence of random variables $X_1, X_2, \ldots, X_n$ the following holds

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

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

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

□

By Lemma 4, we can estimate our joint probability P_{sf} in (3.1) by computing $Pr\{C_1 = 0\}$ and $Pr\{C_d > d-1\}$ for all $d > 1$. 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 occurre 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:

$$W_0 = b \ ; \ W_1 = a$$
$$W_{m+1} = W_m W_{m-1} \quad \text{for } m > 1$$

have $\Theta(n \log n)$ distinct *occurrences* of square subwords. Interestingly, the same applies to the number of different 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 α in T_X such that i and $j=i+|W|$ are consecutive leaves in the subtree of T_X rooted at α and, moreover, $|W(\alpha)|\geq(i-j)$.

It is an easy exercise to implement such a criterion through 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 performance from the average complexity viewpoint.

We devote the remainder of this section to problem (ii). The (primitive rooted) squares in X have consequences on 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 $X=W_1WW_2WW_3\cdots WW_{k+1}$ 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 connection between the auxiliary nodes of T_X and the squares in X is as follows [AP1]. If α is an auxiliary node of the augmented T_X, then there are subwords U and V in X and an integer $k\geq1$ 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, $O(n\log^2 n)$ post-processing 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 [KN] (in short tries). In tries n *independent* keys $X_1, X_2, \ldots, X_n$ are stored, where each key is a (possibly infinite) sequence of symbols over a V-ary alphabet. Note, that in suffix tree the keys $S_1, S_2, \ldots, S_n$ are *dependent* while in the trie it is assumed that the keys are statistically independent. A thorough analysis of tries from the average complexity viewpoint is presented in [SZ1]. In particular, it is proved that the average depth $ED_n = \frac{1}{E}\log n + O(1)$ where E is entropy of the alphabet, that is, $E = -\sum_{i=1}^{V} p_i \ln p_i$. For the average height the following result is known [SZ2, FL]: $EH_n = \frac{2}{\log P^{-1}}\log n + O(1)$ where $P = \sum_{i=1}^{V} p_i^2$. However, for the independent keys it can be proved [SZ1] that the variance of the depth $\mathrm{var}D_n$ is either $O(1)$ for symmetric alphabet (e.g. $\mathrm{var}D_n = 3.507...$ for $V=2$) or $\mathrm{var}D_n = \frac{E_2 - E^2}{E^3}\log n + O(1)$, for the asymmetric tries, where $E_2 = \sum_{i=1}^{V} p_i \ ln^2 p_i$. This implies that asymmetric tries are of an order of magnitude less balanced that the symmetric ones. An open question remains how to evaluate the variance of the depth and the height for suffix trees.

Another issue of some interest is how much the compact version of suffix tree 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).

[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).

[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).

[GL] H. Gerber and S.R. Li, The Occurrence of Sequence Patterns in Repeated Experiments and Hitting Times in a Markov Chain, *Stoch. Proc. Appl.*, 11, 101–108 (1981).

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

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

[LI] S.R. Li, A Martingale Approach to the Study of Occurrence of Sequence Patterns in Repeated Experiments, *Ann. Probab.*, 8, 1171–1176 (1980).

[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, 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, R.J., Lorentz, An O(nlogn) Algorithm for Finding all Repetitions in a String, *Journal of Algorithms*, 422-432 (1984).

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

[NI] P.T. Nielsen, On the Expected Duration of a Search for a Fixed Pattern in Random Data, *IEEE Information Theory*, 702–709 (1973).

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

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

[SZ2] W. Szpankowski, On the Analysis of the Average Height of a Digital Trie: Another Approach, *Purdue University*, CSD TR-646 (1986).

[SZ3] W. Szpankowski, How much on the average is the Patricia trie better, *Proc. of 24-th Annual Allerton Conference on Communications, Control, and Computing*, 314-323 (1986).

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