ELSEVIER Theoretical Computer Science 289 (2002) 1009–1026

Theoretical Computer Science

www.elsevier.com/locate/tcs

Some results on tries with adaptive branching

Yuriy A. Reznik[1]

RealNetworks, Inc., 2601 Elliott Avenue, Seattle, WA 98121, USA

Received November 2001

Abstract

We study a modification of *digital trees* (or *tries*) with *adaptive multi-digit branching*. Such tries can dynamically adjust degrees of their nodes by choosing the number of digits to be processed per lookup. While we do not specify any particular method for selecting the degrees of nodes, we assume that such selection can be accomplished by examining the number of strings remaining in each sub-tree, and/or estimating parameters of the input distribution. We call this class of digital trees *adaptive multi-digit tries* (or *AMD-tries*) and provide a preliminary analysis of their expected behavior in a memoryless model. We establish the following results: (1) there exist AMD-tries attaining a constant expected time of a successful search; (2) there exist AMD-tries consuming a linear (in the number of strings inserted) amount of space; (3) both constant search time and linear space usage can be attained if the (memoryless) source is symmetric. We accompany our analysis with a brief survey of several known types of adaptive trie structures, and show how our analysis extends (and/or complements) previous results. © 2002 Elsevier Science B.V. All rights reserved.

Keywords: Digital tree; Trie; Adaptive branching; Distributive partitioning; N-tree; LC-trie; Average case analysis of algorithms; Asymptotic analysis

1. Introduction

Digital trees (also known as *radix search trees*, or *tries*) represent a convenient way of organizing alphanumeric sequences (strings) of variable lengths that facilitates their fast retrieving, searching, and sorting (cf. [10,15,25]). If we designate a set of n distinct strings as $S=\{s_1,\dots,s_n\}$, and assume that each string is a sequence of symbols from

E-mail address: yreznik@real.com (Y.A. Reznik).

[1] On leave from the Institute of Mathematical Machines and Systems of the National Academy of Sciences of Ukraine.

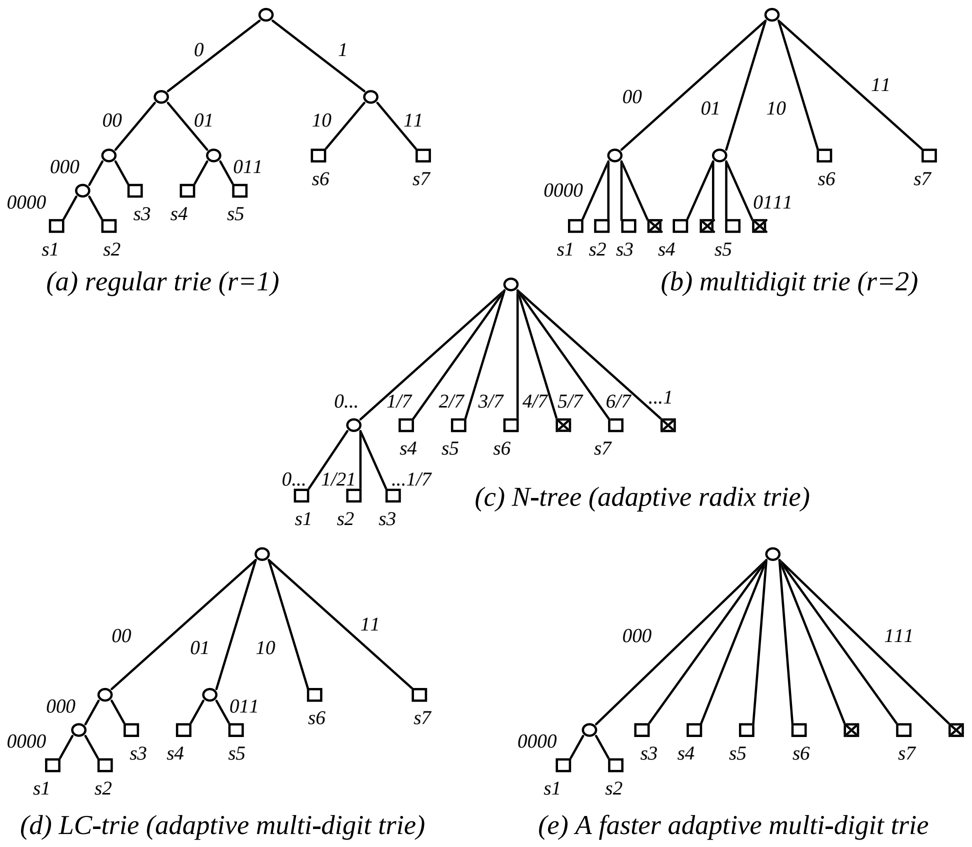

0
1
00
01
10
11
000
011
0000
s1
s2
s3
s4
s5
s6
s7
(a) regular trie (r=1)
0111
(b) multidigit trie (r=2)
0...
1/7
2/7
3/7
4/7
5/7
6/7
...1
1/21
...1/7
(c) N-tree (adaptive radix trie)
(d) LC-trie (adaptive multi-digit trie)
000
111
(e) A faster adaptive multi-digit trie

Fig. 1. Examples of tries built from 7 binary strings: $s1=0000\ldots$, $s2=0001\ldots$, $s3=0010\ldots$, $s4=0100\ldots$, $s5=0110\ldots$, $s6=100\ldots$, $s7=110\ldots$.

a finite alphabet $\Sigma=\{\alpha_1,\ldots,\alpha_v\}$, $|\Sigma|=v$, then a trie $T(S)$ over S can be constructed recursively as follows. If $n=0$, the trie is an *empty external node*.[2] If $n=1$ (i.e. S has only one string), the trie is an *external node* containing a pointer to this single string in S. If $n>1$, the trie is an *internal node* containing v pointers (or *branches*) to the child tries: $T(S_1),\ldots,T(S_v)$, where each set S_i $(1\leqslant i\leqslant v)$ contains suffixes of all strings from S that begin with a corresponding first symbol. For example, if a string $s_j=u_jw_j$ (u_j is a first symbol, and w_j is a string containing the remaining symbols of s_j), and $u_j=\alpha_i$, then the string w_j will go into S_i. Thus, after all child tries $T(S_1),\ldots,T(S_v)$ are recursively processed, we arrive at a tree-like data structure, where the original strings $S=\{s_1,\ldots,s_n\}$ can be uniquely identified by the paths from the root node to non-empty external nodes (see Fig. 1a).

[2] In a simpler (and somewhat more conventional) definition, the case $n=0$ corresponds to an empty trie. While both definitions are essentially equivalent (see, e.g. Knuth [14, 2.3.4.5]), we prefer to use the former as it simplifies transition into a multi-digit case.

A simple extension of the above structure is obtained by allowing more than one symbol of the input alphabet to be used for branching. Thus, if a trie uses some constant number of symbols $r \geqslant 1$ per lookup, then the effective branching coefficient is equal to v^r, and we essentially deal with a trie built over an alphabet Σ^r (see Fig. 1b). To underscore the fact that branching is implemented using multiple input symbols (digits) at a time, such tries are sometimes called *multi-digit tries* [2].

The behavior of regular tries is thoroughly analyzed (cf. [15,5,9,11,23,27]). For example, it has been shown that the expected number of nodes examined during a successful search in a v-ary trie is asymptotically $\log n/h + \mathrm{O}(1)$, where h is the entropy of a process used to produce n input strings. The expected size of such trie is asymptotically $nv/h + \mathrm{O}(1)$. These estimates are known to be correct for a rather large class of stochastic processes, including *memoryless*, *Markovian*, and ψ-mixed models [23,26]. In many special cases (e.g. when a source is memoryless, or Markovian) the complete characterizations (expectation, variance, and higher moments) of these parameters have been obtained, and their precise (up to O(1) term) asymptotic expansions have been deducted (cf. [13,27,12]).

Much less known are modifications of tries that use *adaptive branching*. That is, instead of using nodes of some fixed degree (e.g., matching the cardinality of an input alphabet), *adaptive tries* select branching factor dynamically, from one node to another. Perhaps the best-known example of this idea is *sorting by distributive partitioning*, due to Dobosiewitz [6]. This algorithm (also known as an *N-tree* [8]) selects the degrees of nodes to be equal exactly the number of strings inserted in the sub-tries they originate. For example, an N-tree displayed on Fig. 1c, contains seven strings overall, and therefore its root node has seven branches. In turn, its first branch receives three strings to be inserted, and thus the N-tree creates a node with three additional branches, and so on.

While N-trees have extremely appealing theoretical properties (thus, according to Tamminen [28], and most recently, Mahmoud et al. [19], N-trees attain a constant $(\mathrm{O}(1))$ expected search time, and use a linear $(\mathrm{O}(n))$ amount of memory), there are several important factors that limit their practical usage. The main problem is that the N-tree is not a *dynamic data structure*. It is more or less suitable for a multi-pass construction when all n strings are given, but an addition or removal of a string in an existing structure is rather problematic. In the worst case, such an operation may involve the reconstruction of the entire tree, making the cost of its maintenance extremely high. Somewhat more flexible is a $B - b$ parameterized version of the distributive partitioning proposed in [7]. This algorithm selects the branching factors to be equal n/b $(b \geqslant 1)$, and split child nodes only when the number of strings there becomes larger than B. When both B and b equal one, we have a normal N-tree, however, when they are large, the complexity of updates can be substantially reduced.

Unfortunately, this does not help with another problem in adaptive tries. Since the degrees of nodes are being selected dynamically, N-trees cannot use simple per-symbol lookups. Instead, they must implement dynamic conversions from one size alphabet to another, or even treat the input strings as real numbers in $[0,1)$ [6,8]. In either scenario, the transition between levels in N-trees (and their $B - b$ variants) is much slower than one in regular tries, and combined with the complexity of the maintenance, it creates serious constrains for the usage of N-trees in practice.

In this paper, we focus on another implementation of adaptive branching that promises to be (at least partially) free from the above mentioned shortcomings. We are trying to create an adaptive version of *multi-digit tries* by allowing the number of digits processed by their nodes (parameter r) to be selected dynamically. Due to the lack of an established term, we will call these structures *adaptive multi-digit tries* (or *AMD-tries*). To cover a wide range of possible implementations of such tries, we (at least initially) do not specify any particular algorithm for selecting the degrees of their nodes. Instead, we assume that such selection can be accomplished by examining the number of strings remaining in each sub-tree, and estimating parameters of the input distribution, and attempt to study the resulting class of data structures. The goal of this research is to find performance bounds (in both search time and space domains) attainable by AMD-tries, and deduce several particular implementations that can be of interest in practice.

Somewhat surprisingly, there were only few attempts to explore the potential of adaptive multi-digit branching in the past. Perhaps the only studied implementation in this class is a *level-compressed trie* (or *LC-trie*) of Andersson and Nilsson [2]. The heuristic for selecting the degrees of multi-digit nodes in a LC-trie is very simple: it combines the (v-ary) levels of the corresponding regular trie until it reaches the first external node (see Fig. 1d). It has been shown (cf. [20,22]) that in a memoryless model such algorithm produces nodes with the expected number of digits equal $Er = \log(k)/\eta_{-\infty} + O(\log\log k)$, where k is the number of strings inserted in a sub-trie originated by the node, and $\eta_{-\infty} = -\log p_{\min}$, $p_{\min} = \min\{\Pr(\alpha_i)\}$. When the memoryless source is symmetric, the expected search time in a LC-trie is only $O(\log^* n)$, however, it grows as $O(\log\log n)$ in the asymmetric case [20].

Note, that in a general case, the degrees of nodes in AMD-tries do not have to be constrained by the last complete (i.e. without empty external nodes) levels in the corresponding regular tries. Moreover, in order to support a dynamic construction, the nodes in AMD-tries must include some additional "sparse" levels (i.e. levels containing empty external nodes). This way the resize of a multi-digit node (e.g. due to insertion of new strings) can be effectively delayed until all its empty leaves are replaced by new (non-empty) child tries [21]. Observe that such a strategy (inclusion of "sparse" levels) also reduces the number of levels remaining to be processed, thus making the resulting AMD-trie faster (see Fig. 1e). However, an overly aggressive inclusion of "sparse" levels will also lead to an increased memory usage, which brings us back to the questions of what are the performance bounds that can be attained by AMD-tries in both time- and space-domains, and whether (or when) they can be competitive with other search structures.

In our analysis, we show that in a memoryless model there exist implementations of AMD-tries attaining the constant ($O(1)$) complexity of a successful search regardless of the symmetry of the source. Moreover, if the source is symmetric, such tries can be implemented in linear ($O(n)$) amount of space. Compared to an N-tree, an equally fast AMD-trie appears to have a larger memory usage, however, it is a much more suitable scheme for dynamic implementation, and combined with the benefits of fast multi-digit processing, it promises to be a structure of choice for many practical situations.

This paper is organized as follows. In Section 2, we will provide some basic definitions and present our main results. In Section 3, we will introduce a set of recurrent expressions necessary for the analysis of AMD-tries, and sketch the proofs of our theorems. In Section 4, we will demonstrate how to use our theoretical results to analyze and design new practical algorithms for construction of AMD-tries, and present various experimental results. Finally, in our concluding remarks, we emphasize the limitations of our present study and show several possible directions for future research.

2. Definitions and main results

In our analysis, we only consider AMD-tries built over strings from a binary alphabet $\Sigma = \{0, 1\}$, but the extension to any finite alphabet is straightforward. We also limit our study to a situation when n strings to be inserted in a trie are generated by a *memoryless* (or *Bernoulli*) source (cf. [3]). In this model, symbols of the alphabet Σ occur independently of one another, so that if x_j is the jth symbol produced by this source, then for any j: $\Pr\{x_j = 0\} = p$, and $\Pr\{x_j = 1\} = q = 1 - p$. If $p = q = \frac{1}{2}$, such source is called *symmetric*, otherwise it is *asymmetric* (or *biased*). In our analysis, we will also use the following additional parameters of memoryless sources:

$$\begin{aligned} h &= -p \log p - q \log q; \\ h_2 &= p \log^2 p + q \log^2 q; \\ \eta_\infty &= -\log p_{\max}; \\ \eta_{-\infty} &= -\log p_{\min}; \end{aligned} \tag{1}$$

where: h is the (Shannon's) entropy of the source (cf. [3]), η_∞, and $\eta_{-\infty}$ are special cases of the Rényi's k-order entropy (cf. [26]): $\eta_k = -1/k \log(p^{k+1} + q^{k+1})$, $p_{\max} = \max\{p, q\}$, and $p_{\min} = \min\{p, q\}$. Observe that η_∞, $\eta_{-\infty}$, and h have the following relationship:

$$\eta_\infty \leqslant h \leqslant \eta_{-\infty} \tag{2}$$

(the equality is attained when the source is symmetric, however, in the asymmetric case, these bounds are rather weak).

In this paper, we will evaluate two major the performance characteristics of the AMD-tries: the expected *time of a successful search* and the expected *amount of memory* used by a trie built over n strings. To estimate the search time we can use the expected *depth* D_n or the expected *external path length* (i.e. the combined length of paths from root to all non-empty external nodes) C_n in a trie: $D_n = C_n/n$. To estimate the size of a trie we will use the expected number of its *branches* B_n. Note that the last metric is slightly different from one used for the regular tries (cf. [15,9]). In that case, it was sufficient to find the expected *number of internal nodes* A_n in a trie. However, since internal nodes in adaptive tries have different sizes, we need to use another parameter (B_n) to take into account these differences as well.

As we have mentioned earlier, we allow AMD-tries to use an arbitrary (but not random) mechanism for selecting the degrees of their nodes. However, in a Bernoulli model, there are only two parameters that can affect the expected outcome of such selection applied to all sub-tries in a trie: (a) the number of strings inserted in a sub-trie, and (b) the parameters of source distribution. Thus, without significant loss of generality or precision, we can assume that the node size selection logic can be presented as a (integer-valued) function:

$$r_n := r(n, p), \tag{3}$$

where n indicates the number of strings inserted in the corresponding sub-trie, p is the probability of 0 in the Bernoulli model, and r_n is the number of digits assigned to the resulting (multi-digit) node.

We are now ready to present our main results regarding the expected behavior of AMD-tries. The following theorem establishes the existence of AMD-tries attaining the constant expected time of a successful search.

Theorem 1. *There exist AMD-tries such that*:

$$1 < \xi_1 \leqslant C_n/n \leqslant \xi_2 < \infty, \tag{4}$$

where ξ_1, ξ_2 are some positive constants. The numbers of digits r_n processed by nodes in such tries satisfy:

$$r_n \geqslant \frac{-1}{\eta_{-\infty}} \log(1 - \mathrm{e}^{-\log \xi_2/(n-1)}) = \frac{1}{\eta_{-\infty}} (\log n - \log\log \xi_2) + \mathrm{O}\left(\frac{1}{n}\right) \tag{5}$$

and

$$r_n \leqslant \frac{-1}{\eta_{\infty}} \log(1 - \mathrm{e}^{-\log \xi_1/(n-1)}) = \frac{1}{\eta_{\infty}} (\log n - \log\log \xi_1) + \mathrm{O}\left(\frac{1}{n}\right). \tag{6}$$

Notice that the above conditions are based on parameters η_∞ and $\eta_{-\infty}$, which, in turn, can be considered as bounds for the entropy h of the source (2). This may suggest that there should be a more accurate way to estimate the degrees of nodes needed to attain a certain performance (and vice versa), expressed in terms of the entropy h of the source. The following theorem answers this conjecture in affirmative.

Theorem 2. *Let*

$$r_n^* = \frac{\log n}{h} \tag{7}$$

and assume that the actual degrees of nodes in an AMD-trie are selected to be sufficiently close to r_n^*:

$$|r_n - r_n^*| = \mathrm{O}(\sqrt{r_n^*}). \tag{8}$$

Then the complexity of a successful search in such trie is (*with* $n \to \infty$):

$$C_n/n = \Phi^{-1}\left(\frac{r_n - r_n^*}{\sigma\sqrt{\log n}}\right)\left(1 + \mathrm{O}\left(\frac{1}{\sqrt{\log n}}\right)\right), \tag{9}$$

where

$$\Phi(x) = \frac{1}{\sqrt{2\pi}}\int_{-\infty}^{x} \mathrm{e}^{-t^2/2}\,\mathrm{d}t, \tag{10}$$

is the distribution function of the standard normal distribution (*cf.* [1]), *and*

$$\sigma^2 = \frac{(h_2 - h^2)}{h^3}. \tag{11}$$

Now we will try to evaluate AMD-tries from the memory usage perspective. Theorem 3 establishes bounds for AMD-tries that attain the linear (with the number of strings inserted) size.

Theorem 3. *There exist AMD-tries such that*:

$$1 < \xi_1 \leqslant B_n/n \leqslant \xi_2 < \infty \tag{12}$$

where ξ_1, ξ_2 *are some positive constants. The numbers of digits* r_n *processed by nodes in such tries satisfy*:

$$\begin{aligned} r_n &\geqslant \psi(n, \xi_2, \eta_{-\infty}) \\ &= \frac{1}{\eta_{-\infty}}(\log n - \Delta(n, \kappa_{-\infty}, \xi_2) - \log\log \xi_2) + \mathrm{O}\left(\frac{\log\log n}{\log n}\right) \end{aligned} \tag{13}$$

and

$$\begin{aligned} r_n &\leqslant \psi(n, \xi_1, \eta_{\infty}) \\ &= \frac{1}{\eta_{\infty}}(\log n - \Delta(n, \kappa_{\infty}, \xi_1) - \log\log \xi_1) + \mathrm{O}\left(\frac{\log\log n}{\log n}\right), \end{aligned} \tag{14}$$

where: $\psi(n, \xi, \eta)$ *is the exact solution of*

$$\xi n = 2^{\psi(n,\xi,\eta)}(1 - \mathrm{e}^{-\psi(n,\xi,\eta)\eta})^{1-n}, \tag{15}$$

$\kappa_{\infty} = \eta_{\infty}/\log 2$, $\kappa_{-\infty} = \eta_{-\infty}/\log 2$, *and*

$$\Delta(n, \kappa, \xi) = \log\left(1 + \frac{\kappa \log n - \log(n\kappa)}{\kappa \log \xi}\right). \tag{16}$$

Observe that in the symmetric case ($p = q = \frac{1}{2}$), we have $\kappa_{\infty} = \kappa_{-\infty} = 1$, and thus, the term (16) becomes zero: $\Delta(n, 1, \xi) = 0$. The resulting bounds for (13) and (14) will be identical to one we have obtained in the Theorem 1, Eqs. (5), and (6), and thus, we have the following conclusion.

Corollary 1. *In the symmetric Bernoulli model, an AMD-trie can attain a constant expected depth $D_n = C_n/n = \mathrm{O}(1)$ and have a linear size $B_n = \mathrm{O}(n)$ at the same time.*

Unfortunately, in the asymmetric case the situation is not the same. Thus, if $\kappa \neq 1$, the term $\Delta(n,\kappa,\xi)$ can be as large as $\mathrm{O}(\log\log n)$, and the conditions for the constant search time (5), (6) may not hold. Actually, from the analysis of LC-tries [20], we know that the use of nodes with $r \leqslant \log(n)/\eta_{-\infty}$ leads to the expected search time of $\mathrm{O}(\log\log n)$, and it is not yet clear if this bound can be improved considering the other linear space implementations of the AMD-tries.

We will continue the discussion of the results of our Theorems in Section 4, where we will also describe several possible implementations of AMD-tries and present experimental results.

3. Analysis

In this section, we will introduce a set of recurrent expressions for various parameters of *AMD-tries*, derive some necessary asymptotic expansions, and will sketch the proofs of our main theorems. For the purposes of compact presentation of our intermediate results, we will introduce the following two variables. Parameter x_n will represent some characteristic (e.g. path length or number of nodes) of an *AMD-trie* containing n strings, and parameter y_n will represent a corresponding characteristic of the root node of this trie that contributes to x_n. The following mapping to the standard trie parameters is obvious:

$$\begin{array}{c|ccc} x_n & A_n & B_n & C_n \\ y_n & 1 & 2^{r_n} & n \end{array} \tag{17}$$

where: A_n is the average number of internal nodes, B_n is the average number of branches, C_n is the external path length in an *AMD-trie* containing n strings.

Using the above notation, we can now formulate the following recurrent relationship between these parameters of *AMD-tries*.

Lemma 1. *The properties x_n and y_n of an AMD-trie in a Bernoulli model satisfy*:

$$\begin{aligned} x_n &= y_n + \sum_{k=2}^{n} \binom{n}{k} \sum_{s=0}^{r_n} \binom{r_n}{s} (p^s q^{r_n-s})^k (1 - p^s q^{r_n-s})^{n-k} x_k; \\ x_0 &= x_1 = y_0 = y_1 = 0; \end{aligned} \tag{18}$$

where: *n is the total number of strings inserted in an AMD-trie, and the rest of parameters are as defined in* (3) *and* (17).

Proof. Consider a root node of order r_n (bits) in an *AMD-trie*. If $n < 2$ the result is 0 by definition. If $n \geqslant 2$, the property of trie x_n should include the property of its root node y_n plus the sum of properties of all the child tries. It remains to enumerate child tries and estimate probabilities of them having $2 \leqslant k \leqslant n$ strings inserted. This can be done using the following technique. Let i be one of the 2^{r_n} possible bit-patterns of length r_n, and let $\pi_i = \Pr\{\mathrm{Bin}_{r_n}(s_j) = i\}$ $(0 \leqslant i < 2^{r_n}$, $\sum \pi_i = 1$ be the probability of having a single string s_j, such that its first r_n bits $(\mathrm{Bin}_{r_n}(s_j))$ match the pattern i. Let also assume that the probability π_i is the same for all strings s_j, $0 \leqslant j < n$. Then, the probability that any k strings match pattern i is $\binom{n}{k}\pi_i^k(1-\pi_i)^{n-k}$, and therefore, the contribution of the pattern i to the property x_n can be expressed as $\sum_{k \geqslant 2}\binom{n}{k}\pi_i^k(1-\pi_i)^{n-k}x_k$. Now, it remains to scan all the 2^{r_n} possible patterns to obtain the complete expression for x_n. Recall, that the actual strings we are inserting in the trie are produced by a binary memoryless source (with probabilities $p, q = 1-p$ of symbols 0, and 1 correspondingly). Therefore, the probabilities of the r_n-bit sequences produced by this source depend only on the number of 0's (s) they contain: $\pi(s) = p^s q^{r_n - s}$. Also, given s zeros, the total number of patterns yielding this probability is $\binom{s}{r_n}$. Combining all these formulas, we arrive at

$$\sum_{s=0}^{r_n}\binom{r_n}{s}\sum_{k=2}^{n}\binom{n}{k}(p^s q^{r_n-s})^k(1-p^s q^{r_n-s})^{n-k}x_k,$$

which, after the addition of y_n yields the claimed expression (18). □

It should be stressed that due to the dependency upon an unknown parameter r_n the rigorous analysis of a recurrent expression (18) appears to be a very difficult task. It is not clear for example, if it is possible to convert (18) to a closed form (for any of the parameters involved). Moreover, the attempts to use some particular formulas (or algorithms) for r_n can actually make the situation even more complicated.

The analysis of (18) that we provide in this paper is based on a very simple approach, which nevertheless is sufficient to evaluate some special cases in the behavior of these algorithms. Thus, most of our theorems claim the existence of a solution in linear form, and we use (18) to find bounds for r_n such that the original claim holds. We perform the first step in this process using the following lemma.

Lemma 2. *Let ξ_1 and ξ_2 be two positive constants $(1 < \xi_1 \leqslant \xi_2 < \infty)$, such that*

$$\xi_1 n \leqslant x_n \leqslant \xi_2 n. \tag{19}$$

Then, the recurrent expression (18) *implies that*:

$$\xi_1 \leqslant y_n / f(n, p, r_n) \leqslant \xi_2, \tag{20}$$

where

$$f(n, p, r_n) = n\sum_{s=0}^{r_n}\binom{r_n}{s}p^s q^{r_n-s}(1-p^s q^{r_n-s})^{n-1}. \tag{21}$$

Proof. Consider the upper bound in (19) first, and substitute x_k with $k\xi_2$ in the right side of (18). This yields:

$$\begin{aligned} x_n &\leqslant y_n + \sum_{s=0}^{r_n} \binom{r_n}{s} \sum_{k=2}^{n} \binom{n}{k} (p^s q^{r_n - s})^k (1 - p^s q^{r_n - s})^{n-k} \xi_2 k \\ &= y_n + \xi_2 n \sum_{s=0}^{r_n} \binom{r_n}{s} p^s q^{r_n - s} (1 - (1 - p^s q^{r_n - s})^{n-1}) \\ &= y_n + \xi_2 n - \xi_2 f(n, p, r_n). \end{aligned} \tag{22}$$

Now, according to (19), $x_n \leqslant \xi_2 n$, and combined with (22), the upper bound holds only if $y_n - \xi_2 f(n, p, r_n) \leqslant 0$. Hence $\xi_2 \geqslant y_n / f(n, p, r_n)$, and repeating this procedure for the lower bound ($x_n \geqslant \xi_1 n$), we arrive at formula (20), claimed by the lemma. □

The next step in our analysis is to find bounds for the sum (21) that would allow us to separate r_n. The following lemma summarizes a few such results.

Lemma 3. *Consider a function $f(n, p, r_n)$ defined in* (21). *The following hold*:

$$f(n, 1/2, r_n) = n(1 - 2^{-r_n})^{n-1}, \tag{23}$$

$$f(n, p, r_n) \leqslant n(1 - \mathrm{e}^{-r_n \eta_{-\infty}})^{n-1} \leqslant n \mathrm{e}^{-(n-1)\mathrm{e}^{-r_n \eta_{-\infty}}}, \tag{24}$$

$$f(n, p, r_n) \geqslant n(1 - \mathrm{e}^{-r_n \eta_{\infty}})^{n-1} \geqslant n \mathrm{e}^{-(n-1)\mathrm{e}^{-r_n \eta_{\infty}}/(1 - \mathrm{e}^{-r_n \eta_{\infty}})}. \tag{25}$$

In addition, if $n, r_n \to \infty$, and $|r_n - \log n/h| = \mathrm{O}(\sqrt{\log n/h})$, then asymptotically:

$$f(n, p, r_n) = n\Phi\left(\frac{r_n - \log n/h}{\sigma\sqrt{\log n}}\right)\left(1 + \mathrm{O}\left(\frac{1}{\sqrt{\log n}}\right)\right), \tag{26}$$

where $\Phi(x)$ and σ are as defined in Theorem 2 (7)–(10).

Proof. The equality for the symmetric case (23) is obtained by direct substitution. Two other bounds (24) and (25) can be obtained with the help of the following estimate:

$$\mathrm{e}^{-r_n \eta_{-\infty}} = p_{\min}^{r_n} \leqslant p^s q^{r_n - s} \leqslant p_{\max}^{r_n} = \mathrm{e}^{-r_n \eta_{\infty}},$$

where $p_{\max}$, $p_{\min}$, η_{∞}, and $\eta_{-\infty}$ are as defined in (1). We apply these bounds to the right factor in the sum (21):

$$(1 - \mathrm{e}^{-r_n \eta_{\infty}})^{n-1} \leqslant (1 - p^s q^{r_n - s})^{n-1} \leqslant (1 - \mathrm{e}^{-r_n \eta_{-\infty}})^{n-1},$$

so that it can be separated. The remaining part of the sum (21) converges: $\sum \binom{r_n}{s} p^s q^{r_n - s} = 1$. The additional (right-side) inequalities in (24) and (26) are due to [1, Eq. 4.2.29]:

$$\mathrm{e}^{-x/(1-x)} \leqslant 1 - x \leqslant \mathrm{e}^{-x}, \quad 0 \leqslant x < 1.$$

The derivation of an asymptotic expression (26) is a complicated task, and here we will rather refer to an Example 8.19 in a book of Szpankowski [26] which discusses it in detail. A somewhat different approach to solving it has also been described in [17,18]. □

Now using the results of the above lemmas, we can sketch the proofs of our main theorems. Consider the problem of evaluating the behavior of *AMD-tries* from the expected complexity of a successful search perspective first.

We can use the recurrent expression (18) where, according to (17), we can substitute x_n with a parameter of the external path length C_n, and y_n with n. Observe that the bounds for C_n (4) claimed by the Theorem 1 and condition (19) in Lemma 2 are equivalent, and the combination of (20) with inequalities (24), and (25) result in

$$\xi_2 \geqslant n/f(n,p,r_n) \geqslant (1-e^{-r_n\eta_{-\infty}})^{1-n} \tag{27}$$

and

$$\xi_1 \leqslant n/f(n,p,r_n) \leqslant (1-e^{-r_n\eta_{\infty}})^{1-n}. \tag{28}$$

The solution of (27) with respect to r_n yields:

$$r_n \geqslant \frac{-1}{\eta_{-\infty}}\log(1-e^{-\log\xi_2/(n-1)}) = \frac{1}{\eta_{-\infty}}(\log n - \log\log\xi_2) + O\left(\frac{1}{n}\right), \tag{29}$$

which is exactly the condition (5) claimed by the Theorem 1. The corresponding solution for (28) leads to condition (6) in Theorem 1.

The result of the Theorem 2 follows directly from (20), (21), and asymptotic expression (26).

To evaluate the expected size an *AMD-trie* in memory we will use a very similar technique. Consider a recurrent expression (18) with x_n substituted by the expected number of branches B_n, and y_n with 2^{r_n}. Following the Theorem 3 and Lemma 2, we assume that B_n is bounded according to (12), and using (20) combined with (24) and (25) we obtain:

$$\xi_2 \geqslant 2^{r_n}/f(n,p,r_n) \geqslant 2^{r_n}n^{-1}(1-e^{-r_n\eta_{-\infty}})^{1-n} \tag{30}$$

and

$$\xi_1 \leqslant 2^{r_n}/f(n,p,r_n) \leqslant 2^{r_n}n^{-1}(1-e^{-r_n\eta_{\infty}})^{1-n}. \tag{31}$$

Compared to (27) and (28) these appear to be slightly more complicated expressions, which do not yield simple exact solutions for r_n. To find an asymptotic (for large n) solution of (30), we write:

$$r_n \geqslant \frac{-1}{\eta_{-\infty}}\log(1-e^{-\log(\xi_2 n 2^{-r_n})/(n-1)}) = \frac{1}{\eta_{-\infty}}(\log n - \log\log(\xi_2 n 2^{-r_n})) + O\left(\frac{1}{n}\right),$$

where after some transformations we arrive at:

$$r_n \geqslant \frac{1}{\log 2}(\log n + \log\xi_2 + \kappa_{-\infty}^{-1}W_{-1}(-\kappa_{-\infty}\xi_2^{-\kappa_{-\infty}}n^{1-\kappa_{-\infty}})) + O\left(\frac{1}{n}\right),$$

where $\kappa_{-\infty} = \eta_{-\infty}/\log 2$, and $W_{-1}(x)$ is a branch $W(x) \leqslant -1$ of the *Lambert W* function: $W(x)e^{W(x)} = x$ [4]. To perform further simplifications we can use the following asymptotic expansion of $W_{-1}(x)$ (cf. [4]):

$$W_{-1}(x) = \log(-x) - \log(-\log(-x)) + O\left(\frac{\log(-\log(-x))}{\log(-x)}\right),$$
$$-e^{-1} < x < 0,$$

which, after some algebra yields the second part of inequality (13) claimed by the Theorem 3. The expression for the lower bound (14) is obtained in essentially the same way.

4. Applications and experimental results

In this section, we will show how to use our theorems to analyze and design new algorithms for construction of AMD-tries. We will complement our analysis with a number of experiments measuring the actual time- and space-performance characteristics of these algorithms.

First, we would like to show how to use our Theorems 1 and 3 for evaluation of the performance bounds of some practical implementations of AMD-tries.

As an example, we will consider a modification of LC-trie algorithm recently proposed by Nilsson and Tikkanen [21]. They observed that a j-digit node in an AMD-trie has several easy to track parameters, such as the numbers of pointers to empty e_j, external x_j, and internal a_j nodes in the trie (see Fig. 2). The total number of pointers in such a node is $e_j + x_j + a_j = 2^j$. Consequently, a *density* of this node (i.e. a ratio of pointers to all non-empty nodes to all pointers in a j-digit node) is

$$\rho(j) = \frac{x_j + a_j}{2^j} = 1 - \frac{e_j}{2^j}. \quad (32)$$

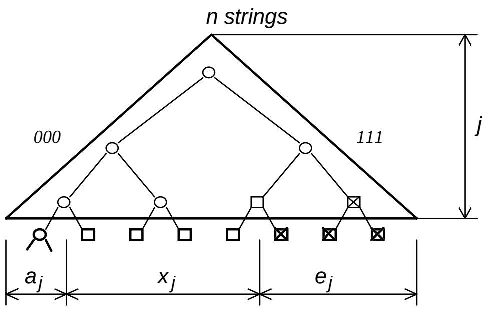


Fig. 2. Simple parameters of a j-digit node that can be used by an AMD-trie construction algorithm: e_j—the number of empty nodes, x_j—the number of external nodes, and a_j—the number of the remaining internal nodes/subtries.

The algorithm of Nilsson and Tikkanen selects the number of digits r_n for its nodes such that their densities are constrained by some positive constants ρ_1 and ρ_2:

$$r_n \in \{j : 0 < \rho_1 \leqslant \rho(j) \leqslant \rho_2 \leqslant 1\}. \tag{33}$$

When both $\rho_1 = \rho_2 = 1$ the resulting trie is an LC-trie. However, when $\rho_1 < \rho_2$ the resulting trie can be much faster, and we will be interested to learn if such algorithm (33) can attain a constant expected search time.

Using the techniques discussed in Section 3, we can show that:

$$1 - \rho_1 \geqslant e_{r_n}/2^{r_n} = 2^{-r_n} \sum_{s=0}^{r_n} \binom{r_n}{s} (1 - p^s q^{r_n - s})^n \geqslant (1 - \mathrm{e}^{-r_n \eta_{-\infty}})^n$$

and consequently,

$$r_n \leqslant \frac{-1}{\eta_{-\infty}} \log(1 - \mathrm{e}^{\log(1-\rho_1)/n}) = \frac{1}{\eta_{-\infty}} (\log n - \log(-\log(1 - \rho_1))) + \mathrm{O}\left(\frac{1}{n}\right).$$

This is clearly the opposite of the condition (5) of Theorem 1, which means that in the asymmetric case, the density-constrained construction (33) is not sufficient to attain the constant expected time of a successful search.

Second, we will now try to use the result of Theorem 2 to design a few new algorithms for construction of AMD-tries with constant expected search time.

According to the formula (9) of Theorem 2, such algorithms must choose nodes with $r_n \sim r_n^* = \log n/h$. The main challenge with using this formula directly is that, in practice, we do not have any a priori information about the source (so we cannot calculate h). A possible solution here is to estimate parameters of the source dynamically. Such an algorithm can, for example, count frequencies of symbols as they appear in input strings, and employ the standard techniques, such as Krichevsky–Trofimov (KT) [16] or Laplace [24] probability estimates. An obvious drawback of such a mechanism is the complexity associated with updating frequency counts and entropy calculations.

Fortunately, using the natural statistics in tries we can estimate the values r_n^* in another, remarkably simple way. From the analysis of regular tries we know that the depths of the non-empty external nodes are asymptotically *normally distributed*, with mean at $\log n/h + \mathrm{O}(1)$ and variance $\sigma^2 \log n + \mathrm{O}(1)$ [11]. Therefore, we immediately have three simple (mean-, mode-, and median-based) algorithms for selecting r_n:

$$r_n^{(1)} = \frac{1}{n} \sum_{i \geqslant 1} i z_i, \tag{34}$$

$$r_n^{(2)} = \min\left\{ j : z_j = \max_{1 \leqslant i < \infty} \{z_i\} \right\}, \tag{35}$$

$$r_n^{(3)} = \min\left\{ j : \sum_{1 \leqslant i \leqslant j} z_i \geqslant \frac{n}{2} \right\}, \tag{36}$$

where z_i is the number of external nodes on level i in the original (e.g. binary) trie.

The last formula (36) is of particular interest. Observe, that its sum

$$\sum_{1 \leqslant i \leqslant j} z_i = x_j,$$

where x_j is the number of external nodes in a combined j-digit node (see Fig. 2). This quantity also means the total number of strings that have been uniquely identified by this j-digit node. For this reason we can call a ratio

$$\xi(j) = \frac{x_j}{n} \tag{37}$$

a *selectivity* of this node. We can use this parameter $\xi(j)$ to present (36) in a somewhat more flexible form (which we need to minimize the number of updates in the trie):

$$r_n \in \{j : 0 < \xi_1 \leqslant \xi(j) \leqslant \xi_2 \leqslant 1\}, \tag{38}$$

where ξ_1 and ξ_2 are some positive constants.

To simplify our future notation, we will call a trie constructed according to the above algorithm (38) a *selectivity-constrained* AMD trie, and similarly, a trie constructed according to Nilsson and Tikkanen algorithm (33), a *density-constrained* AMD-trie.

In addition to the above described algorithms, in our experimental study we will also evaluate the following two schemes. The first algorithm is a *logarithmic trie* representing a direct attempt to construct a multi-digit version of an N-tree:

$$r_n = \lceil \log_2 n \rceil,$$

which is of some interest since we know that the original N-tree is both O(1)-fast and O(n)-large. The second algorithm is an offline implementation of a formula (7) suggested by our Theorem 2:

$$r_n = \lceil \log(n)/h \rceil$$

(h is known or estimated based on symbols in all n strings before trie construction is started).

On Figs. 3 and 4 we present the results of experimental evaluation of the expected successful search time and expected memory usage of several implementations of AMD tries.

To build our tries we used computer-generated sequences of binary digits for symmetric ($p = 0.5$) and asymmetric ($p = 0.25$) cases. To be able to identify even extremely slowly growing functions (e.g. $\log^* n$) we allowed the number of strings n in tries to grow from 2 to 10^5. At each point (each fixed value for n) we generated 10^3 different tries (using the same source model), and calculated their average depths and relative sizes.

Observe that in the symmetric case (see Fig. 3a), LC-trie is the only structure which depth is clearly increasing with n (and its rate well matches the theoretic estimate $O(\log^* n)$ [2]). The depths of the other AMD tries are fluctuating in the range $(1.5, 2.5)$ with no visible drift upward, which suggests that they are likely O(1)-fast.

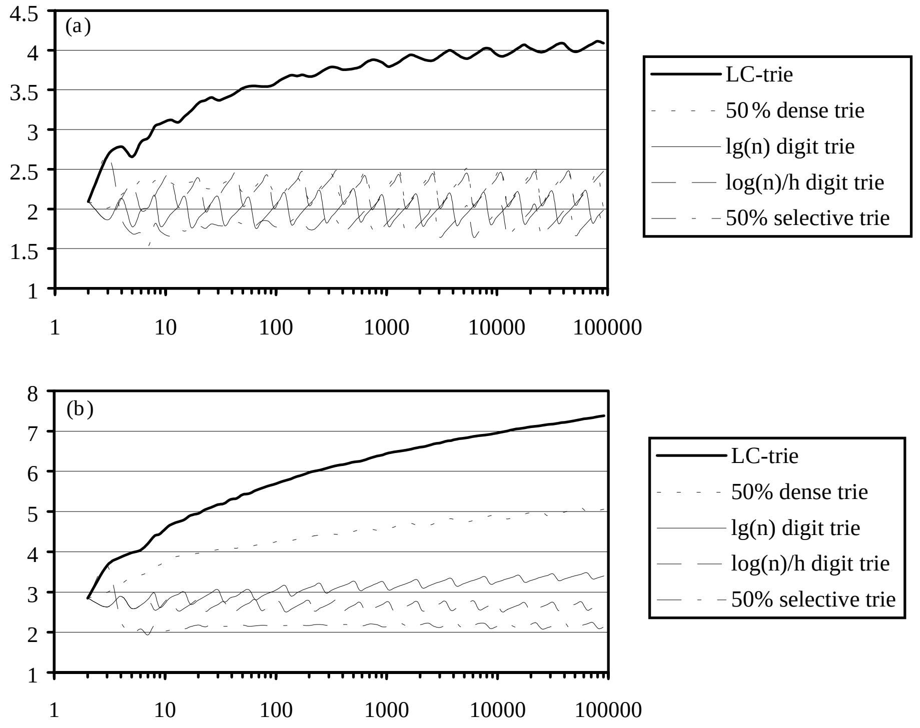
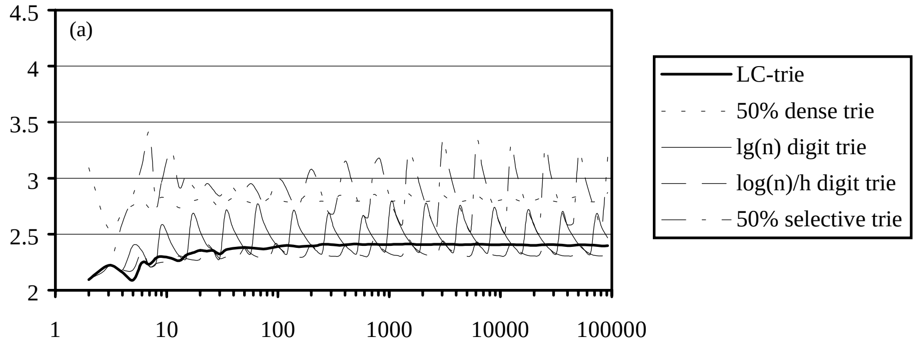


Fig. 3. Average depths C_n/n of several modifications of AMD tries when (a) memoryless source is symmetric ($p=0.5$), and (b) when the source is asymmetric ($p=0.25$). Axis x represents the number of strings n inserted in tries.

The situation is quite different in the asymmetric case (see Fig. 3b). Here, both LC- and density-constrained tries are growing rapidly, $\log_2 n$-digit tree also has some tendency to grow, but at much slower rate (likely $O(\log^* n)$), while both selectivity-constrained and our Theorem 2-based tries just fluctuate in the range $(1.8, 2.8)$, which suggests that they are O(1)-fast (which is what we claimed in our Theorem 2).

Analyzing the results for the expected relative size (see Fig. 4a), we can conclude that all of our modifications are likely $O(n)$—large when the source is symmetric. In the asymmetric case (see Fig. 4b), however, both of our O(1)—fast implementations tend to grow very rapidly (at least with $B_n = O(n^{\log_2/h})$ rate), with a selectivity-constrained variant being the worst (which is to be expected, since the numbers r_n it produces for various sub-tries only converge (at least in probability) to the needed quantity $\log(n)/h$, instead of being always less than or equal to it). The $\log_2 n$-digit trie seem to grow much slower (again, likely at $B_n/n = O(\log^* n)$ rate), while the relative sizes of both LC- and sparsity-constrained tries are fluctuating between constants.

These results can be seen both as illustrations confirming the main statements of our theorems, and even more importantly, as a demonstration of the enormous spectrum



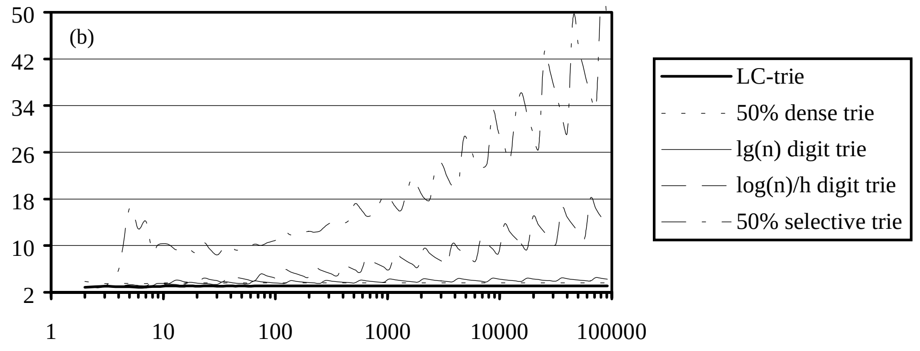


Fig. 4. Average relative sizes B_n/n of several modifications of AMD tries when (a) memoryless source is symmetric ($p=0.5$), and (b) when the source is asymmetric ($p=0.25$). Axis x represents the number of strings n inserted in tries.

of possible solutions available to engineers designing data structures with given space–time-performance tradeoffs.

5. Concluding remarks

We absolutely believe that *AMD-tries* have great potential that should be explored in practice. From this perspective, our conclusion that they can attain O(1) expected search time, while preserving most of the benefits of the regular tries, should be a good starting point.

At the same time, *AMD-tries* pose several interesting theoretical problems, and in this area, our results just scratch the surface. The technique that we used in our analysis was only sufficient to produce (rather coarse) bounds for the expected characteristics of *AMD-tries*, and the derivation of their exact expressions (including lower-magnitude terms and oscillating components) remains an open (and difficult) problem.

Another interesting problem is to find parameters of an *AMD-trie* that uses the minimum possible amount of memory. Thus, analyzing the set of expressions in the proof of the Theorem 3 we can conjecture that such a trie exists, but finding its actual parameters requires a more solid analytical framework.

Acknowledgements

The author wishes to express his gratitude to Prof. W. Szpankowski from Purdue University for many long and fruitful discussions and encouragement in starting this research. The author is also obliged to one of the referees whose very careful reading and suggestions helped to eliminate many inaccuracies and led to a much better presentation of the results.

References

[1] M. Abramowitz, I. Stegun, Handbook of Mathematical Functions, Dover, New York, 1972.
[2] A. Andersson, S. Nilsson, Improved behaviour of tries by adaptive branching, Inform. Process. Lett. 46 (1993) 295–300.
[3] T.M. Cover, J.M. Thomas, Elements of Information Theory, Wiley, New York, 1991.
[4] R.M. Corless, G.H. Gonnet, D.E.G. Hare, D.J. Jeffrey, D.E. Knuth, On the Lambert W function, Adv. Comput. Math. 5 (1996) 329–359.
[5] L. Devroye, A note on the average depths in tries, SIAM J. Comput. 28 (1982) 367–371.
[6] W. Dobosiewitz, Sorting by distributive partitioning, Inform. Process. Lett. 7 (1) (1978) 1–6.
[7] W. Dobosiewitz, The practical significance of DP sort revisited, Inform. Process. Lett. 8 (4) (1979) 170–172.
[8] G. Ehrlich, Searching and sorting real numbers, J. Algorithms 2 (1981) 1–14.
[9] P. Flajolet, R. Sedgewick, Digital search trees revisited, SIAM J. Comput. 15 (1986) 748–767.
[10] E. Fredkin, Trie memory, Comm. ACM 3 (1960) 490–500.
[11] P. Jacquet, M. Régnier, Trie Partitioning Process: Limiting Distributions, Lecture Notes in Computer Science, Vol. 214, Springer, New York, 1986, pp. 196–210.
[12] P. Jacquet, W. Szpankowski, Analysis of digital trees with Markovian dependency, IEEE Trans. Inform. Theory 37 (1991) 1470–1475.
[13] P. Kirschenhofer, H. Prodinger, Some Further Results on Digital Search Trees, Lecture Notes in Computer Science, Vol. 229, Springer, New York, 1986, pp. 177–185.
[14] D. Knuth, The Art of Computer Programming, Fundamental Algorithms, Vol. 1, Addison-Wesley, Reading, MA, 1968.
[15] D. Knuth, The Art of Computer Programming, Sorting and Searching, Vol. 3, Addison-Wesley, Reading, MA, 1973.
[16] R.E. Krichevsky, V.K. Trofimov, The performance of universal encoding, IEEE Trans. Inform. Theory 27 (1981) 199–207.
[17] G. Louchard, The Brownian motion: a neglected tool for the complexity analysis of sorted tables manipulations, RAIRO Theoret. Inform. 17 (1983) 365–385.
[18] G. Louchard, Digital search trees revisited, Cahiers du CERO 36 (1995) 259–273.
[19] H. Mahmoud, P. Flajolet, P. Jacquet, M. Régnier, Analytic variations on bucket selection and sorting, Acta Inform. 36 (9/10) (2000) 735–760.
[20] S. Nilsson, Radix sorting and searching, Ph.D. Thesis, Department of Computer Science, Lund University, 1996.
[21] S. Nilsson, M. Tikkanen, Implementing a Dynamic Compressed Trie, Proc. Second Workshop on Algorithm Engineering (WAE'98), Saarbruecken, Germany, 1998, pp. 25–36.

[22] B. Pittel, Asymptotic growth of a class of random trees, Ann. Probab. 18 (1985) 414–427.
[23] B. Pittel, Paths in a random digital tree: limiting distributions, Adv. Appl. Probab. 18 (1986) 139–155.
[24] J. Rissanen, Complexity of strings in the class of Markov sources, IEEE Trans. Inform. Theory 32 (1986) 526–532.
[25] R. Sedgewick, P. Flajolet, An Introduction to the Analysis of Algorithms, Addison-Wesley, Reading, MA, 1996.
[26] W. Szpankowski, Average Case Analysis of Algorithms on Sequences, Wiley, New York, 2001.
[27] W. Szpankowski, Some results on V-ary asymmetric tries, J. Algorithms 9 (1988) 224–244.
[28] M. Tamminen, Analysis of N-trees, Inform. Process. Lett. 16 (3) (1983) 131–137.