Dichotomic Selection on Words: A Probabilistic Analysis

Ali Akhavi
GREYC (Normandie Université, Unicaen, EnsiCaen, Cnrs), 14000, Caen, France
ali.akhavi@unicaen.fr

Julien Clément
GREYC (Normandie Université, Unicaen, EnsiCaen, Cnrs), 14000, Caen, France
julien.clement@unicaen.fr

Dimitri Darthenay
GREYC (Normandie Université, Unicaen, EnsiCaen, Cnrs), 14000, Caen, France
dimitri.darthenay@gmail.com

Loïck Lhote
GREYC (Normandie Université, Unicaen, EnsiCaen, Cnrs), 14000, Caen, France
loick.lhote@unicaen.fr

Brigitte Vallée
GREYC (Normandie Université, Unicaen, EnsiCaen, Cnrs), 14000, Caen, France
brigitte.vallee@unicaen.fr

Abstract

The paper studies the behaviour of selection algorithms that are based on dichotomy principles. On the entry formed by an ordered list L and a searched element $x \notin L$, they return the interval of the list L the element x belongs to. We focus here on the case of words, where dichotomy principles lead to a selection algorithm designed by Crochemore, Hancart and Lecroq, which appears to be "quasi-optimal". We perform a probabilistic analysis of this algorithm that exhibits its quasi-optimality on average.

2012 ACM Subject Classification Theory of computation → Data structures design and analysis; Theory of computation → Randomness, geometry and discrete structures; Theory of computation → Sorting and searching; Theory of computation → Pattern matching; Mathematics of computing → Combinatorics on words; Mathematics of computing → Discrete mathematics

Keywords and phrases dichotomic selection, text algorithms, analysis of algorithms, average case analysis of algorithms, trie, suffix array, lcp–array, information theory, numeration process, sources, entropy, coincidence, analytic combinatorics, depoissonization techniques

Digital Object Identifier 10.4230/LIPIcs.CPM.2019.19

Funding Projects: DynA3S, CoDys, MetaConc (ANR), AleaEnAmSud (STICAmSud).

Acknowledgements We thank Pablo Rotondo for many interesting discussions. We also thank anonymous reviewers for pointing us to bibliographic references.

1 Introduction

The dichotomic search [18] is one of the most basic tool for locating the position of a target value x within a sorted list of n elements. This scheme, that has a classical "divide-and-conquer" flavour, is a good algorithmic compromise in many situations, because it is straightforward to implement and nonetheless guarantees a $\Theta(\log n)$ number of comparisons between x and the n elements of the list.

General context. In this paper, we are interested in the case when the list and the target values are *words*, that are emitted by a *source*. Then, the comparison between two words is the usual (lexicographic) comparison, and the performance of the dichotomic selection on words is measured by the total number of *symbol comparisons*. More precisely, we study algorithms that are designed in the book of Crochemore, Hancart, and Lecroq [7, chapter 4]. In this context, the list is fixed and many target values are searched: it is then natural to consider the list sorted (thanks to a precomputation step), and use dichotomic principles[1]. Such algorithms are notably the basis for efficient implementations of searching in the suffix array [19, 2, 22, 16, 21], which is a widely used index structure in text algorithmics [14, 7].

We focus on a particular algorithm described in [7, Chapter 4], and called here CLEVER-DICHO-SELECT. The authors explain there that the sorting precomputation is not actually sufficient to build an efficient search procedure: one needs to use a supplementary structure, called the LCP–array in [17, 20, 15, 12, 10] that stores the length of longest common prefixes between consecutive strings of the list[2]. The precomputation step which determines the LCP–array has a worst-case complexity linear in the total number of symbols of all words in the list. With these two precomputation steps at hand (the sorting precomputation and the LCP–array precomputation), the technique becomes very efficient for searching for a string of length m in a list of n strings with $O(m + \log n)$ symbol comparisons (in the worst-case). Remark that the usual algorithm that reads the word x and uses a *lexicographic tree* or *trie* for finding the longest common prefix of the target value x within the list of strings (which is the minimal information needed to locate the position of x) yields $O(m)$ symbol comparisons.

Motivations. There are three motivations for such a work.

(a) We wish to perform a precise analysis of variants of the dichotomic search algorithm on strings in the following framework described with four features : (i) the performance is measured in terms of the number of *symbol comparisons*; (ii) we adopt a *probabilistic* or *average-case* point of view which is significantly different from the worst-case one; (iii) we are interested in the *asymptotics*, when the number n of strings become large, and accordingly choose to deal with infinite words (to be detailed later); (iv) finally, we wish to determine the precise constants involved in the analysis.

(b) A main motivation is also to better understand how the structure of data influence the performance of algorithms. In many real applications, such data, of highly composite nature, are often aggregates of elementary symbols, (such as bits, bytes, characters...). It is then legitimate to consider such data as *words*, and the elementary cost is now the comparisons between symbols. This present work follows a series of recent works of the authors [27, 6, 4] where they revisited the *probabilistic* analysis of a panel of some classic sorting and selecting algorithms from this point of view (for instance Quicksort). Here we take the same perspective and wish to analyze the general dichotomic strategy, from the basic algorithm to the more clever version of [7, Chapter 4], when it operates on complex data.

(c) Our probabilistic modelling first involves the concept of a source, that describes how the input words (the elements of the list L, and the searched element x) are emitted. Such a source denoted by $\mathcal{M}$ extends the notion of a numeration process (see [25]), and

[1] Other strategies are possible apart from the dichotomic principles for the problem searching in a sorted list of strings (see [1, 11]).

[2] The used structure is in fact the LCP–table which is a slight variation of the usual LCP–array, as we will describe later.

any (infinite) word emitted by the source $\mathcal{M}$ can be viewed as the expansion of a real number of the unit interval in "base" $\mathcal{M}$. Besides this source $\mathcal{M}$, that emits the input words, there appears also a second *implicit* source (the regular binary source $\mathcal{B}$), which models the dichotomy process. The interplay of these two sources is then central in our analyses. Even though such an interplay between two sources often arises in numeration contexts, and is well studied there (see for instance [3, 8]), notably in relation to the various Changing Base algorithms, it came as a surprise for us that it also arises in our analyses.

Main results. In the context of our analysis, defined by the four features described in (a), we analyze in Theorem 9 the CLEVER-DICHO-SELECT algorithm. We first analyze the precomputation step, that builds (and stores) the LCP–table (an integer array of length $2n+1$ which is a slight variation of the usual LCP–array), and prove that the mean complexity cost for building the table is of order $\Theta(n)$. Then, we analyse the mean complexity cost of the algorithm itself and prove that it is of order $\Theta(\log n)$. It is thus quasi-optimal[3] *on average*. We precisely study the constants hidden in the Θ, and relate them to the main characteristics of the source, and, in particular, to the interplay between the two sources described in (c): the *input source* that emits the words, and the *dichotomic source* which models the dichotomic process.

Methodology. We first exhibit the main costs for evaluating the time complexity of the algorithm, that are not highlighted in the previous work [7]. These costs are read on the two main structures, the dichotomic tree $D(L)$ which underlies the partitioning process, together with *trie* $T(L)$ built on the list L (which was not explicit in [7]). We also introduce another strategy for computing the LCP–table related to an original parameter called `dic`. Then, we perform the average-case analysis of the main costs: we mainly deal with what we call *costs with toll* on the trie $T(L)$ (defined in Eq. (5)), and we adapt the analytic combinatorics methodology for such an analysis, as described both in [5] and [26].

Plan of the paper. Section 2 first recalls the general framework of the dichotomic strategy. Then, Section 3 focuses on the selection problems on words; it presents the CLEVER-DICHO-SELECT algorithm, and exhibits its main costs of interest. Finally, Section 4 is devoted to the probabilistic analysis of these costs, and proves the quasi-optimality (on average) of the CLEVER-DICHO-SELECT algorithm.

2 Basic Dichotomic Selection

Let us consider an ordered set $\mathcal{X}$. The problem SELECT (L, x) takes as input a list L of n distinct elements of $\mathcal{X}$, together with an element $x \notin L$, and determines the rank of x in L, namely the integer i for which x will be the i-th smallest element in the sorted set $L \cup \{x\}$. When the list $L = (L_1, \ldots, L_n)$ is already sorted, the well-known dichotomic strategy may be used. It deals with the *extended list* $\overline{L} := (L_0, L_1, \ldots, L_n, L_{n+1})$, where two *guards* are added to the initial list L, a *strict infimum* L_0 and a *strict supremum* L_{n+1} satisfying the strict inequalities $L_0 < y < L_{n+1}$ for any $y \in \mathcal{X}$. The dichotomy principle uses the function $\text{mid} : (b, e) \mapsto \lfloor (b+e)/2 \rfloor$, and determines the *rank* of x in $L \cup \{x\}$, i.e., the index $i \in [1\,..\,n+1]$ for which $L_{i-1} < x < L_i$. The algorithm DICHO-SELECT is described in Fig. 1 (left).

[3] compared to the bound $\Theta(\log n)$ obtained *on average* when dealing with a precomputed trie.

```
DICHO-SELECT (L, x)
b := 0
e := n + 1
while b + 1 < e do
    m := ⌊(b + e)/2⌋
if  L_m < x then
    b := m
else
    e := m
return e
```

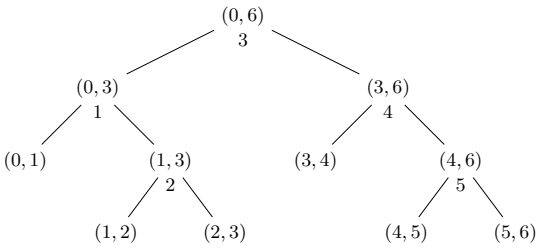


Figure 1 The DICHO-SELECT algorithm (left). The dichotomic tree D_n ($n = 5$) (right).

With each cardinality n, the dichotomy strategy associates a binary tree D_n, called the *dichotomic tree* (Fig. 1, right). Such a tree describes the partitioning process on any ordered list of cardinality n, with indices in $[1\,..\,n]$, and deals with extended indices in $[0\,..\,n+1]$.

▶ **Definition 1** (Dichotomic tree). *The* dichotomic tree D_n *associated with a cardinality* n *is a binary tree, with* n *internal nodes and* $n+1$ *external nodes. Each node is labelled by a pair* (b, e) *with* $0 \le b < e \le n+1$. *Moreover,*

- *if* $b + 1 < e$, *the node labelled by* (b, e) *is* internal*: it contains the index* $m = \mathrm{mid}(b, e)$ *and has two children, labelled by* (b, m) *(on the left) and* (m, e) *(on the right);*
- *if* $b + 1 = e$, *the node labelled by* (b, e) *is* external*: it has no children.*

An integer pair (c, d) *is dichotomic it appears as a label of a node in* D_n.

When the DICHO-SELECT algorithm is called on a sorted list L of cardinality n, the dichotomic tree D_n is adapted to the list L (and its extended version $\overline{L}$): an internal node (b, e) contains the key L_m with $m = \mathrm{mid}(b, e)$. The tree is thus called the dichotomic tree of the list L, and denoted by $D(L)$. During the execution of DICHO-SELECT on the input (L, x), the tree $D(L)$ is used as a binary search tree, and a comparison is performed at node (b, e) between the key x and the key L_m with $m = \mathrm{mid}(b, e)$.
The sequence of visited nodes forms a path in $D(L)$, that is called the *execution path* $\Pi(L, x)$. It leads to an external node with label $(i-1, i)$ such that $L_{i-1} < x < L_i$. The part $\Pi(L, x)$ that excludes the last (external) node and thus only gathers its internal nodes, is called the *internal execution path* and denoted by $\underline{\Pi}(L, x)$. The length of this path, denoted by $|\underline{\Pi}(L, x)|$ and defined as the number of its nodes, is a central parameter of DICHO-SELECT(L, x):

▶ **Lemma 2.** *The total number* $G(L, x)$ *of key comparisons performed by* DICHO-SELECT *on the input* (L, x) *is equal to the length* $|\underline{\Pi}(L, x)|$ *of the internal execution path* $\underline{\Pi}(L, x)$ *of* DICHO-SELECT *in* $D(L)$ *on input* x. *When the cardinality* n *of* L *belongs to* $[2^{p-1}\,..\,2^p - 1]$, *the length of the path* $\underline{\Pi}(L, x)$ *satisfies* $|\underline{\Pi}(L, x)| \in \{p-1, p\}$. *When* $n = 2^p - 1$, *one has* $|\underline{\Pi}(L, x)| = p$, *for any* x.

3 Dichotomic selection for words

3.1 General context

The DICHO-SELECT algorithm is straightforwardly adapted to words, as shown in [7]. The book describes the framework that appears in practical issues, where one deals with a list L of finite words (called strings), that may be possibly prefixes of other strings inside L, and where the searched string x may belong to L. This makes the pseudo code of [7] rather subtle and sometimes difficult to understand.

As we focus on asymptotic features, when both the cardinality of the list, and the length of strings tend to ∞, we are led to the case when the keys (both the elements of the list L and the searched element x) are infinite words. Furthermore, we suppose that all words in L are distinct and the word x does not belong to L. This makes the precise structure of algorithms more readable and their main parameters more apparent: this leads to a clearer asymptotic complexity analysis, notably on average.

We will design various algorithms of dichotomic flavour that adapt the DICHO-SELECT algorithm to the context of infinite words, built over some finite ordered alphabet Σ. We now deal with two different orderings, the partial prefix order $\preceq$ (defined on prefixes), and the lexicographic order $\leq$ (defined on infinite words). For two finite words w, w' on the alphabet Σ, we denote $w \preceq w'$ (or $w' \succeq w$) if w is a prefix of w'. We denote by $\Gamma(x, y)$ the *longest common prefix* between two words x and y and by $\gamma(x, y)$ its length, here called the *coincidence*[4] between x and y. Then, the number of symbol comparisons needed to compare the words x and y is equal to $\gamma(x, y) + 1$, and the main complexity parameter is the total number of symbol comparisons performed by the algorithm.

3.2 The trie structure

The *trie* structure (see [13, 23]) is the main tree structure for solving problems on words. Fig. 2 gives an example of a trie built on a list L with seven words on $\Sigma := \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$.

▶ **Definition 3.** *Let L be a list of words on the ordered alphabet $\Sigma := \{a_1, a_2, \ldots, a_r\}$. The* trie *$T(L)$ is defined as follows*

- *if $L = \emptyset$, then $T(L) = \emptyset$; if $L = \{w\}$, then $T(L)$ is an* external node *which contains w;*
- *for $|L| \geq 2$, $T(L)$ is an* internal node *with r subtrees $T(L \setminus a_1), \ldots, T(L \setminus a_r)$, where $L \setminus u$ is the list formed with words of L that start with the symbol u stripped of u; the edge between the root and the i-th subtree is labelled with the symbol a_i.*

The trie only maintains the minimal prefix set $P(L)$ of symbols that is needed to distinguish all the elements of L (written underlined in blue in Fig. 2). Without any prior knowledge about the set of the words, this is the most efficient tool for comparing (and sorting) words.

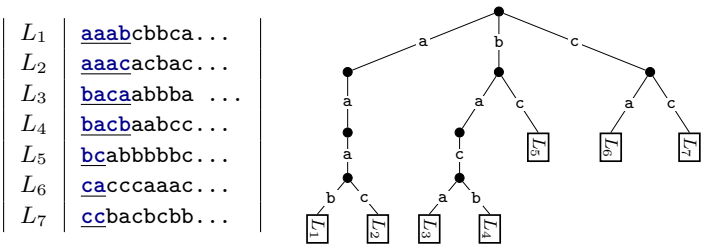


Figure 2 On the left, a list L with seven words, with its minimal prefix set $P(L)$ (underlined). On the right, the trie $T(L)$.

▶ **Proposition 4.** *The minimal number of symbol comparisons needed to solve the selection problem* SELECT *(L, x) on words (without prior knowledge on the input (L, x)) is exactly the length of the branch $B(L, x)$ of the trie $T(L \cup \{x\})$ leading to the external node associated with x.*

[4] This parameter is usually denoted as LCP(x, y) and reads as (length of) the least common prefix. As we both deal with the prefix itself, and its length, we prefer the notations Γ and γ.

The trie structure has been deeply analyzed in various contexts (see for instance [18, 9, 24]). In a quite general probabilistic framework of a source $\mathcal{M}$ with entropy $h(\mathcal{M})$ (defined in [25, 6] and later recalled in Section 4.1), the following holds, in terms of number of symbol comparisons, and on average, when the cardinality n of the list L tends to ∞ (see [5]):

- the asymptotic cost of *building the trie* $T(L)$ is equivalent to $[1/h(\mathcal{M})]\, n \log n$;
- the asymptotic size of the minimal prefix set $P(L)$ is equivalent to $[1/h(\mathcal{M})]\, n \log n$;
- the asymptotic cost $|B(L,x)|$ of *inserting* x in $T(L)$ is equivalent to $[1/h(\mathcal{M})] \log n$.

But, in the present context, the list L is assumed to be ordered. The main questions are now as follows: *Is it possible to take advantage of the ordering of the list L, with a clever use of the dichotomic strategy? Can this be done without explicitly building or storing the trie $T(L)$? Are these dichotomic algorithms close to the lower bound $|B(L,x)|$?*

The CLEVER-DICHO-SELECT algorithm proposed in [7] (and recalled later in Section 3.4) answers all these questions in an affirmative way.

Even though the trie $T(L)$ is not explicitly built during the execution of our further algorithms of type DICHO-SELECT on words, our analyses "read" operations made by the algorithms simultaneously on the two trees, the dichotomic tree $D(L)$ and the trie $T(L)$. With a node labelled by (b,e) in the dichotomic tree $D(L)$, we first consider the two elements L_e and L_b of the list $\overline{L}$ (that may be fictive guards for $e=0$ or $b=n+1$), then the prefix $\Gamma[b,e] := \Gamma(L_b, L_e)$ which is the largest common prefix between L_b and L_e, and finally its length $\gamma[b,e]$. In the case when $b=0$ or $e=n+1$, we let $\Gamma[b,e] := \varepsilon$ (the empty word) and $\gamma[b,e]=0$. This gives rise to the $\Gamma(L)$ tree that is a decoration[5] of the dichotomic tree $D(L)$ and contains at each node (b,e) of $D(L)$ the prefix $\Gamma[b,e]$ (see Fig. 5, right). This prefix $\Gamma[b,e]$ also identifies a node of the trie $T(L)$ and the associated branch in $T(L)$, from the root to this node.

A path $\Pi(L,x)$ of $D(L)$ leading to an external node of label $(i-1,i)$ gives rise to the branch labelled with $\Gamma[i-1,i]$ in the trie $T(L)$. Then, inserting x in $T(L)$ yields the branch $B(L,x)$ in the trie $T(L \cup \{x\})$. The sequence of prefixes $\Gamma[m,x] := \Gamma(L_m, x)$ is increasing (for the prefix order $\preceq$) along the path $\Pi(L,x)$, and the strategy of the algorithm defines how $\Gamma[m,x]$ is computed along $\Pi(L,x)$. We describe in the following three different strategies; we explain why the first two ones, defined in Section 3.3, are not efficient, and how they can be adapted in Section 3.4 to lead to the CLEVER-DICHO-SELECT Algorithm that is proven quasi-optimal in Section 4.2.

3.3 First two variants of dichotomic search on strings

Substituting the key-comparison "$L_m < x$" in DICHO-SELECT(L,x) with the red block of Fig. 3 (left) implementing the comparison between words gives rise to the algorithm WORD-DICHO-SELECT(L,x) described in Fig. 3 (left). The WORD-DICHO-SELECT algorithm appears to be quite naive, since it recomputes from scratch the prefix $\Gamma[m,x]$ along the path $\Pi(L,x)$. It does not use the important following property due to the ordering of the list:

▶ **Lemma 5.** *For a node of $\underline{\Pi}(L,x)$ with label (b,e) and index $m := \mathrm{mid}(b,e)$, and for any $x \in [L_b, L_e]$, the inequality holds: $\Gamma[m,x] \succeq \min(\Gamma[b,x], \Gamma[e,x]) = \Gamma[b,e]$.*

Lemma 5 leads to a more efficient algorithm, the DICHO-SELECT-WMIN algorithm described in Fig. 3 (right), where the added blue lines maintain along the path $\underline{\Pi}(L,x)$ the

[5] This means that $\Gamma(L)$ has the same structure as $D(L)$ with some extra information at nodes.

current coincidences $\ell_b := \gamma[b, x]$ and $\ell_e := \gamma[e, x]$ between the searched key x and the two ends of the interval $[L_b, L_e]$ it belongs to. However, the DICHO-SELECT-WMIN algorithm is not optimal. The following indeed holds along the execution path $\underline{\Pi}(L, x)$:

- At node (b, e), the algorithm deals with prefixes w that satisfy $\Gamma[b, e] \preceq w \preceq \Gamma[m, x]$;
- At the successor node with label (b', e'), the strict inequality $\Gamma[b', e'] \prec \Gamma[m, x]$ may hold; thus, the algorithm may go *backwards* along the branch $B(L, x)$, *repeating* the same symbol comparisons;

```
WORD-DICHO-SELECT (L, x);
b := 0 ; e := n + 1;
while b + 1 < e
      do m := mid(b, e) ;
ℓ := 0 ;
while x[ℓ] = L_m[ℓ] do
      ℓ := ℓ + 1;
if  L_m[ℓ] < x[ℓ] then
      b := m
else
      e := m
return e
```

```
DICHO-SELECT-W-MIN (L, x);
b := 0 ; e := n + 1 ;
ℓ_b := 0; ℓ_e := 0;
while b + 1 < e do
      m := mid(b, e);
ℓ := min(ℓ_b, ℓ_e);
while x[ℓ] = L_m[ℓ] do
      ℓ := ℓ + 1 ;
if  L_m[ℓ] < x[ℓ] then
      b := m; ℓ_b := ℓ ;
else
      e := m; ℓ_e := ℓ ;
return e
```

Figure 3 The two algorithms: the WORD-DICHO-SELECT (left), the DICHO-SELECT-WMIN (right).

3.4 A clever version that uses the LCP–table

This is why the authors of [7] propose to precompute the set $\{\gamma[b, e] \mid (b, e) \in D(L)\}$ that only depends on L (and not on x). When stored in an array (see Section 3.5), it is called in [7] the LCP–table and it slightly extends the notion of LCP–array which restricts to pairs of consecutive strings in L. Then, the authors prove that precise comparisons, at each node (b, e) of the path $\underline{\Pi}(L, x)$, between the four values

$$\gamma[b, x],\ \gamma[e, x],\ \gamma[b, m],\ \gamma[e, m], \quad \text{with} \quad m := \mathrm{mid}(b, e)$$

avoids backwards steps and thus redundant symbol comparisons, as it is now stated.

▶ **Lemma 6.** *Consider a node of $\underline{\Pi}(L, x)$ with label (b, e), together with the four coincidences $\ell_b = \gamma[b, x], \ell_e = \gamma[e, x], \gamma[b, m], \gamma[e, m]$. There are five (exclusive) cases in order to determine $\ell_m = \gamma[m, x]$, the successor pair (b', e') (and thus the next pair $(\ell_{b'}, \ell_{e'})$):*

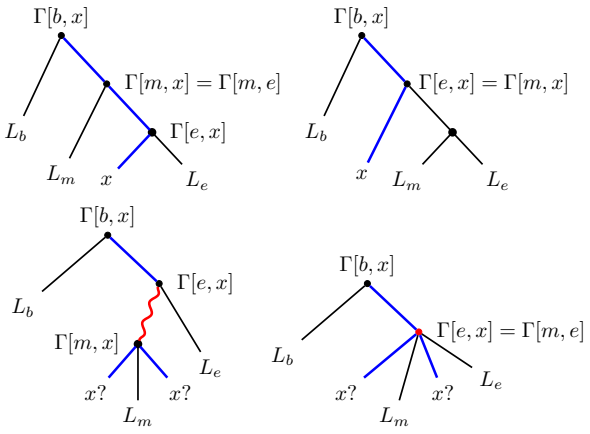
- *In the first four cases, the following holds:*
  (1) *if $\gamma[b, x] > \gamma[b, m]$, then $\ell_m = \gamma[b, m], (b', e') = (b, m)$.*
  (2) *if $\gamma[e, x] > \gamma[e, m]$, then $\ell_m = \gamma[m, e], (b', e') = (m, e)$.*
  (3) *if $\gamma[b, x] < \gamma[b, m]$, then $\ell_m = \ell_b, (b', e') = (m, e)$.*
  (4) *if $\gamma[e, x] < \gamma[e, m]$, then $\ell_m = \ell_e, (b', e') = (b, m)$.*

  The equality $\max(\ell_{b'}, \ell_{e'}) = \max(\ell_b, \ell_e)$ holds; the CLEVER-DICHO-SELECT algorithm does not perform any comparison and does not discover any new symbol;
- *In the remaining case (5), when $\gamma[b, x] = \gamma[b, m]$ and $\gamma[e, x] = \gamma[m, e]$, then a computation is needed for ℓ_m and (b', e'). The inequality $\max(\ell_{b'}, \ell_{e'}) \geq \max(\ell_b, \ell_e)$ holds;*
  The algorithm performs $[\max(\ell_{b'}, \ell_{e'}) - \max(\ell_b, \ell_e) + 1]$ symbol comparisons and discovers $[\max(\ell_{b'}, \ell_{e'}) - \max(\ell_b, \ell_e)]$ new symbols. Case (5) corresponds to a node (b, e) of the path $\underline{\Pi}(L, x)$ (called a forward node*) for which one of the two conditions (i) or (ii) hold*

(i) $\gamma[m,x] > \max(\gamma[b,x], \gamma[e,x])$
(ii) $\gamma[m,x] = \max(\gamma[b,x], \gamma[e,x]) = \max(\gamma[b,m], \gamma[e,m])$

The set of forward nodes (called the forward set*) of* $\underline{\Pi}(L,x)$ *is denoted as* $F(L,x)$.

Fig. 4 (left) leads to an easy proof of Lemma 6, explicitly based on the trie structure $T(L)$: it represents the branch $B(L,x)$ of the trie $T(L)$ (in blue) with the relative possible positions of the three branches that respectively lead to L_b, L_e, L_m. It focuses on the case where $\ell_b \le \ell_e$, which leads to four possible cases (2), (4) $(5i)$, $(5ii)$ of Lemma 6.




```
CLEVER-DICHO-SELECT(L, x);
(b, ℓ_b) := (0, 0); (e, ℓ_e) := (n + 1, 0)
while b + 1 < e do
  m := ⌊(b + e)/2⌋;
  if γ[m, e] < ℓ_e then
    (b, ℓ_b) := (m, γ[m, e])
  else if ℓ_e < γ[m, e] then
    e := m
  else if  γ[b, m] < ℓ_b then
    (e, ℓ_e) := (m, γ[b, m])
  else if ℓ_b < γ[b, m] then
    b := m
  else
    ℓ := max(ℓ_b, ℓ_e)
    while x[ℓ] = L_m[ℓ] do
      ℓ := ℓ + 1
    if L_m[ℓ] < x[ℓ] then
      (b, ℓ_b) := (m, ℓ)
    else
      (e, ℓ_e) := (m, ℓ)
return e.
```

Figure 4 On the left, description of the four cases (2), (4), $(5(i))$, $(5(ii))$ of Lemma 6 that occur when $\gamma[e,x] \ge \gamma[b,x]$. On the right, description of the CLEVER-DICHO-SELECT Algorithm.

Proof of Lemma 6. When the inequality $\Gamma[e,x] \succeq \Gamma[b,x]$ holds, we deal with the branch L_e, and consider the positions (defined by the prefixes $\Gamma[m,e]$ and $\Gamma[e,x]$) where the branches L_m and x are hung on the branch L_e; As the two words L_m and x satisfy (with respect to the lexicographic order) $L_m < L_e$ and $x < L_e$, these branches are hung on the left of the branch L_e: there are three cases for their relative positions:

$$\text{case (2): } \Gamma[e,x] \succ \Gamma[e,m]\ ; \quad \text{case (4): } \Gamma[e,x] \prec \Gamma[e,m]\ ; \quad \text{case (5): } \Gamma[e,x] = \Gamma[e,m]\,.$$

(2) The equality $\Gamma[m,x] = \Gamma[m,e]$ holds. As the branch x holds on the left of L_e, this shows that $x \in [L_m, L_e]$. Then the next (b',e') is (m,e).

(4) The equality $\Gamma[m,x] = \Gamma[e,x]$ holds. As the branch x holds on the left of L_e, this shows that $x \in [L_b, L_m]$. Then the next (b',e') is (b,m).

(5) The two branches x and L_m are hung at the same position on L_e, and both lie on the left of L_e. Then there exists a *red* path, (empty in the case $(5ii)$) which begins at prefix $\Gamma[e,x] = \Gamma[m,e]$ and represents the common suffix between L_m and x (empty in the case $(5ii)$). The comparison between L_m and x has to be performed, and begins at prefix $\Gamma[e,x]$. Remark that the equality $\Gamma[b,x] = \Gamma[b,m]$ also holds in this case. ◀

Then, Lemma 6 gives rise to the CLEVER-DICHO-SELECT algorithm described in Fig. 4 (right), whose complexity is now summarized in the following Proposition.

▶ **Proposition 7.** *Consider a list L, with its precomputed set $\gamma(L)$. Then, the total number of symbol comparisons performed by* CLEVER-DICHO-SELECT *on the input (L, x) is*

$$C(L, x) = |B(L, x)| + |F(L, x)|,$$

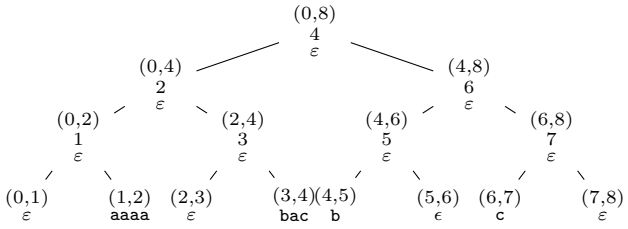
where $B(L, x)$ is the branch that leads to x in the trie $T(L \cup \{x\})$, and $F(L, x)$ is the subset of forward nodes in the internal execution path $\underline{\Pi}(L, x)$ of the dichotomic tree $D(L)$.

3.5 Precomputing the LCP–table

In Section 3.2, the $\Gamma(L)$ tree was defined as a decoration of the tree $D(L)$. Here, we only need the knowledge of on array of length $2n+1$ (Fig. 5, bottom right). This array (called the LCP–table in [7]) contains the value $\gamma[b, e]$ at index $i(b, e) \in [0\,..\,2n]$ defined as: $i(b, e) = b$, if (b, e) is external, and $i(b, e) = n + \text{mid}(b, e)$, if (b, e) is internal.

The computation of the LCP–table proposed in the book [7] is based on a bottom-up strategy, and has a complexity closely related to the path length of the trie $T(L)$, of order $\Theta(n \log n)$ (on average). We present here a new algorithm, based a top-down strategy. When called on the triple $(0, n+1, 0)$ the algorithm BUILD-LCP-TABLE, described in Fig. 5 (left), computes the LCP–table and the number $A(L)$ of symbol comparisons performed clearly involves the difference $\gamma[b, m] + \gamma[m, e] - 2\gamma[b, e]$ at each node $[b, e]$ of $D(L)$.

```
BUILD-LCP-TABLE(b, e, ℓ)
if b = 0 or e = n + 1 then
    ℓ' = 0
else
    ℓ' := ℓ
    while L_b[ℓ'] = L_e[ℓ'] do
        ℓ' := ℓ' + 1
γ[b, e] := ℓ'
if b + 1 < e then m := mid(b, e)
    BUILD-LCP-TABLE(b, m, ℓ')
    BUILD-LCP-TABLE(m, e, ℓ')
```



$$\text{LCP–table}(L) = [0, 4, 0, 3, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0]$$
$$\texttt{Int(a)} = \texttt{Int(aa)} = \texttt{Int(aaa)} = [1, 2],$$
$$\texttt{Int(b)} = [3, 5] = [3, 4] \cup [4, 5], \quad \texttt{Int(c)} = [6, 7]$$
$$\texttt{Dic(a)} = \texttt{Dic(aa)} = \texttt{Dic(aaa)} = \texttt{Dic(ba)} = \{[1, 2]\},$$
$$\texttt{Dic(b)} = \{[3, 4], [4, 5]\}$$

Figure 5 On the left, the algorithm for computing the LCP–table. On the right, the $\Gamma(L)$-tree and the LCP–table related to the list L of Fig. 2, together with some examples for $\texttt{Int}(w)$ and $\texttt{Dic}(w)$.

We now provide an alternative expression for the cost $A(L)$. Consider indeed an internal node of the trie $T(L)$ related to the prefix w and associate with it the largest integer interval $\texttt{Int}(w) := [a, b]$ for which $w = \Gamma[a, b]$. The cardinality of $\texttt{Int}(w)$ is the number N_w of elements of L which begin with w. The interval $\texttt{Int}(w)$ is not *dichotomic*[6] (in general). We

[6] A pair (a, b) is dichotomic if it labels a node in the dichotomic tree $D(L)$.

denote by $\texttt{Dic}(w)$ its decomposition into largest possible dichotomic intervals (see Fig. 5 for some examples), and by $\texttt{dic}(w)$ the cardinality of the decomposition $\texttt{Dic}(w)$. There are now two results: – first, the parameter $\texttt{dic}$ arises as a basic cost for $A(L)$ – second, there is a relation between the two parameters $\texttt{dic}(w)$ and the cardinality N_w .

▶ **Proposition 8.** *The number $A(L)$ of symbol comparisons used by Algorithm* Build-lcp-table *for computing the* lcp*–table for list L is expressed as*

$$A(L) = (2n + 1 - E_n) + \sum_{\substack{w \in T(L) \\ w \neq \varepsilon}} \texttt{dic}(w), \tag{1}$$

and involves the number E_n of nodes on the extreme (left and right) branches of $D(L)$ together with the cardinality $\texttt{dic}(w)$ of the dichotomic decomposition for every internal node w of the trie $T(L)$. One has $E_n = \Theta(\log n)$ and the following estimate holds,

$$\texttt{dic}(w) \leq 2 + 2\log_2 N_w \,. \tag{2}$$

4 Probabilistic analyses of the dichotomic process

The algorithm clever-Dicho-Select algorithm involves thus three parameters:

- The length $|B(L, x)|$ of the branch which leads to x in the trie $T(L \cup \{x\})$; this is a classical parameter, and an instance of a *cost with toll* (see Eq. (5)). It only depends on probabilistic properties of the input source and does not involve the dichotomic process.
- The other two parameters are more difficult to analyze, because they involve both the trie $T(L)$ and the dichotomic tree $D(L)$, and thus two sources (as Section 4.3 will explain).
  - We only propose trivial bounds for the number $|F(L, x)|$ of forward nodes in the dichotomic path $\underline{\Pi}(L, x)$, namely $1 \leq |F(L, x)| \leq |\underline{\Pi}(L, x)|$
  - As Proposition 8 shows, the cost $A(L)$ for computing the lcp–table for L is defined from the cost $\texttt{dic}$. This is *not* a *cost with toll*, but it can bounded from above and below with two costs with toll.

Moreover, the basic structure that underlies the whole dichotomic strategy on the list L is the $\Gamma(L)$ tree, that contains at each node (b, e) the prefix $\Gamma[b, e]$ of length $\gamma[b, e]$. The mean value of the coincidence $\gamma[b, e]$ at a random node (b, e) of depth ℓ is also a central parameter, and we give some hints in Section 4.3 for a possible further study.

4.1 The input source and its parameterization

We consider a source $\mathcal{M}$ on an ordered finite alphabet Σ which will both produce the elements of the list L and the searched word x (called the input source). A source is a (probabilistic) mechanism which emits a symbol from the alphabet Σ, at each discrete time $t = 0, 1, \ldots$. It produces infinite words in $\Sigma^{\mathbb{N}}$ and is defined by the set of probabilities

$$p_w := \Pr[\text{a word begins with the prefix } w], \quad (w \in \Sigma^\star) \,.$$

Figure 6 (top) recalls the parameterization of a source (see also [6]): using, for each k, the equality $\sum_{w \in \Sigma^k} p_w = 1$ and the lexicographic order on Σ^k, builds a sequence of quasi-disjoint intervals $\mathcal{I}_w$, with $|\mathcal{I}_w| = p_w$. When k tends to ∞, each word y is written (almost everywhere) as $y = M(t)$, with $t \in [0, 1]$. The increasing mapping $M : [0, 1] \to \Sigma^{\mathbb{N}}$ maps reals to infinite words. Using an analogy with numeration, we say that the infinite word $M(t)$ is the expansion of the real t in "base" $\mathcal{M}$, and (conversely) the real t is the *parameter* of the word y.

Classical sources are obtained in this general framework: all the memoryless sources – the *regular* ones, for which all the probabilities p_w of the prefixes $w \in \Sigma^k$ are equal, as well as the non regular ones– , but also Markov chains, and finally a class of more correlated sources, as described in [25]. Figure 6 (bottom) describes the parameterization of two regular sources: the binary one (in blue), and the ternary one (in green).

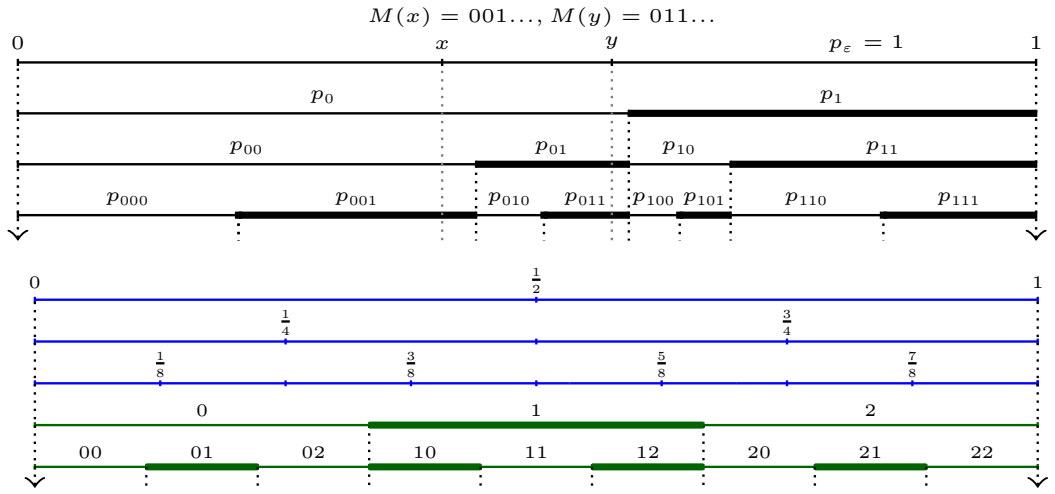


Figure 6 Parameterization of a source and fundamental intervals (top). Two instances of parameterization for regular sources (bottom).

In this context, the Dirichlet generating series of the sources, defined as

$$\Lambda(s) := \sum_{w \in \Sigma^\star} p_w^s, \qquad \Lambda_k(s) := \sum_{w \in \Sigma^k} p_w^s\,, \tag{3}$$

play a prominent role, as highlighted in [25]. The source is said to be *tame* if these functions considered as functions of a complex variable s have good properties near the vertical line $\Re s = 1$ (see [6] and details in the Appendix); it notably possesses an entropy,

$$h(\mathcal{M}) := \lim_{k\to\infty} \frac{1}{k} \sum_{w \in \Sigma^k} p_w \log p_w = \lim_{k\to\infty} \frac{1}{k} \frac{d}{ds} \Lambda_k(s)\Big|_{s=1}\,. \tag{4}$$

The source is said to be *periodic* if the series $\Lambda(s)$ is periodic.
The Dirichlet series of a regular source associated with an alphabet of cardinality b admit nice expressions,

$$\Lambda_k(s) = b^k \cdot b^{-ks} = b^{k(1-s)}, \quad \Lambda(s) = (1 - b^{1-s})^{-1}\,.$$

This shows that a b-regular source is tame and periodic of period $2\pi i/(\log b)$, with an entropy equal to $\log b$.

The intervals that appear at depth k in Figure 6 (top) define the parameterization M. They are called the fundamental intervals (of depth k). They satisfy $\mathcal{I}_w := \{u \mid M(u) \text{ begins with } w\}$. They may be defined via the coincidence γ, as $I_k(t) = \{u \in [0,1] \mid \gamma(M(t), M(u)) \geq k\}$.

Both the elements of the list L and the searched word x will be produced by the source $\mathcal{M}$. We first draw $(n+1)$ real numbers $(t, u_1, \ldots, u_n)$ in the $[0,1]$ interval, in a uniform and independent way; then, we sort $(u_1, u_2, \ldots, u_n)$, and rewrite this n-uple as an ordered n-uple

$(t_1 < t_2 < \cdots < t_n)$; we let $x = M(t)$, and $L_i := M(t_i)$ for $i \in [1 \dots n]$. As the mapping M is increasing, the list L is sorted in the increasing order. The parameter t_i of the word L_i is the i-th smallest element (also called the i-th statistics) of the random n-uple $(u_1, u_2, \dots, u_n)$.

4.2 Probabilistic analysis of the Clever-Dicho-Select algorithm

With a list L and a finite prefix w, we have associated in Section 3.5 the parameter $N_w := N_w(L)$ that is the number of elements of L which begin with w. The internal nodes of the trie $T(L)$ are labelled with prefixes w for which $N_w(L) \geq 2$. The variable N_w plays a central role in the analysis of the trie parameters, notably those that are associated with a *toll* function $x : \mathbb{N} \to \mathbb{N}$ and obtained as a sum taken over the internal nodes:

$$X(L) := \sum_{w | N_w \geq 2} x(N_w)\,. \tag{5}$$

This class of parameters X (called *costs with toll*) contains for instance the path length (for $x(k) = k$) and the number of internal nodes (for $x(k) = 1$.) The asymptotic mean value of such parameters has been deeply studied, notably in [5], and it depends both on the toll and the characteristics of the source, in particular its entropy. Even though the cost $\texttt{dic}(w)$ is *not* a cost with toll, the inequality $1 \leq \texttt{dic}(w) \leq 2 \log_2 N_w + 2$ stated in Proposition8 bounds it with two costs with toll. We then obtain the main result of the paper, for which a sketch of proof is given in the Appendix.

▶ **Theorem 9.** *Consider a* good *input source $\mathcal{M}$ and a random input (L, x) with $|L| = n$ as described in Section 4.1. Then asymptotically:*

- *the mean value B_n of the length $|B(L, x)|$ satisfies $B_n \sim (1/h(\mathcal{M})) \log n$;*
- *the mean value F_n of the cardinality of the forward set satisfies $1 \leq F_n \leq \log_2(n+1)$;*
- *the mean value A_n of the cost $A(L)$ for building the $\gamma(L)$ array satisfies*

$$\frac{n}{h(\mathcal{M})}[1 + P(\log n)] \leq A_n \leq \frac{n}{h(\mathcal{M})}[1 + Q(\log n)] \left[\frac{2}{\log 2} \sum_{k \geq 2} \frac{\log k}{k(k-1)} \right],$$

where P and Q are two periodic functions of very small amplitude which only appear when the source is periodic.

4.3 Probabilistic analysis of the $\Gamma(L)$ tree: the second source

We now consider the probabilistic behaviour of the main structure that underlies the algorithm, namely the $\Gamma(L)$ tree. We focus here on the case of *pure dichotomy* where the cardinality n of the list satisfies $n + 1 = 2^p$ for some integer $p > 0$, and, thus, the dichotomic tree D_n (denoted by $\widehat{D}_p$) is *complete*.

A node (b, e) at depth ℓ in $\widehat{D}_p$ satisfies $b = a\, 2^{p-\ell}$, $e = (a+1)\, 2^{p-\ell}$ for some integer $a \in [0 \,..\, 2^\ell - 1]$. This node is associated in the $\Gamma(L)$ tree with the pair (L_b, L_e), whose parameters t_b and t_e are resp. the b-th and the e-th statistics of the n-uple $(u_1, u_2, \dots, u_n)$, i.e., the elements of respective ranks b and e in the sorted n-uple $(t_1, t_2, \dots, t_n)$. They thus satisfy, with classical results on i-th statistics recalled in the Appendix,

$$[t_b, t_e] \approx [v_b, v_e]\,, \quad \text{with} \quad v_b = b\, 2^{-p} = a\, 2^{-\ell}, \quad v_e := e\, 2^{-p} = (a+1)\, 2^{-\ell}\,. \tag{6}$$

This estimate holds for $p \to \infty$ and any $\ell \leq p$. Then, any node (b, e) of depth ℓ in $\widehat{D}_p$ is (asymptotically for $p \to \infty$) associated with a fundamental interval $[v_b, v_e]$ of depth ℓ of the regular binary source $\mathcal{B}$.

This provides an (asymptotic) geometric interpretation of the coincidence $\gamma[b, e]$ (when $p \to \infty$): this is the largest integer k for which there exists a fundamental interval $\mathcal{I}_w$ of depth k (of the source $\mathcal{M}$) that contains the given fundamental interval $[v_b, v_e]$ of the source $\mathcal{B}$. Such an interplay between two sources –here the sources $\mathcal{M}$ and $\mathcal{B}$– is deeply studied in the context of dynamical systems (see for instance the results of [3, 8]), at least when the depth ℓ tends to ∞. A more precise view of this (asymptotic) interplay between these two sources will be central in a further analysis of the $\Gamma(L)$ tree.

The parameter `Dic` may be also described thanks to the interplay of the two sources: it is indeed related to the decomposition of a fundamental interval of the source $\mathcal{M}$ in terms of fundamental intervals of the source $\mathcal{B}$.

Conclusion. The present study has introduced the $\Gamma(L)$ tree and the parameter `Dic`. The $\Gamma(L)$ tree underlies any algorithm based on dichotomic process, and we feel that the parameter `Dic` plays an important role in this process and should be further studied for itself.

References

1 Arne Andersson, Torben Hagerup, Johan Håstad, and Ola Petersson. Tight Bounds for Searching a Sorted Array of Strings. *SIAM J. Comput.*, 30(5):1552–1578, 2000. doi:10.1137/S0097539797329889.

2 Alberto Apostolico, Maxime Crochemore, Martin Farach-Colton, Zvi Galil, and S. Muthukrishnan. 40 Years of Suffix Trees. *Commun. ACM*, 59(4):66–73, March 2016. doi:10.1145/2810036.

3 Wieb Bosma, Karma Dajani, and Cor Kraaikamp. Entropy quotients and correct digits in number-theoretic expansions. In *Dynamics & stochastics*, volume 48 of *IMS Lecture Notes Monogr. Ser.*, pages 176–188. Inst. Math. Statist., Beachwood, OH, 2006. doi:10.1214/074921706000000202.

4 Julien Clément, James Allen Fill, Thu Hien Nguyen Thi, and Brigitte Vallée. Towards a Realistic Analysis of the QuickSelect Algorithm. *Theory of Computing Systems*, 58(4):528–578, May 2016. doi:10.1007/s00224-015-9633-5.

5 Julien Clément, Philippe Flajolet, and Brigitte Vallée. Dynamical sources in information theory: A general analysis of trie structures. *Algorithmica*, 29(1):307–369, February 2001. doi:10.1007/BF02679623.

6 Julien Clément, Thu-Hien Nguyen-Thi, and Brigitte Vallée. Towards a Realistic Analysis of Some Popular Sorting Algorithms. *Combinatorics, Probability and Computing*, 24(1):104–144, 2015. doi:10.1017/S0963548314000649.

7 Maxime Crochemore, Christophe Hancart, and Thierry Lecroq. *Algorithms on Strings*. Cambridge University Press, New York, NY, USA, 2007.

8 Karma Dajani and Adam Fieldsteel. Equipartition of interval partitions and an application to number theory. *Proc. Amer. Math. Soc.*, 129(12):3453–3460, 2001. doi:10.1090/S0002-9939-01-06299-2.

9 Luc Devroye. A Probabilistic Analysis of the Height of Tries and of the Complexity of Triesort. *Acta Inf.*, 21(3):229–237, October 1984. doi:10.1007/BF00264248.

10 Johannes Fischer. Inducing the LCP-Array. In Frank Dehne, John Iacono, and Jörg-Rüdiger Sack, editors, *Algorithms and Data Structures*, pages 374–385, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

11 Gianni Franceschini and Roberto Grossi. No Sorting? Better Searching! In *45th Symposium on Foundations of Computer Science (FOCS 2004), 17-19 October 2004, Rome, Italy, Proceedings*, pages 491–498, 2004. doi:10.1109/FOCS.2004.43.

12 Simon Gog and Enno Ohlebusch. Fast and Lightweight LCP-array Construction Algorithms. In *Proceedings of the Meeting on Algorithm Engineering & Expermiments*, ALENEX '11, pages 25–34, Philadelphia, PA, USA, 2011. Society for Industrial and Applied Mathematics.

13 G. H. Gonnet and R. Baeza-Yates. *Handbook of Algorithms and Data Structures: In Pascal and C (2Nd Ed.)*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1991.

14 Dan Gusfield. *Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology*. Cambridge University Press, 1997. doi:10.1017/CBO9780511574931.

15 Juha Kärkkäinen, Giovanni Manzini, and Simon J. Puglisi. Permuted Longest-Common-Prefix Array. In Gregory Kucherov and Esko Ukkonen, editors, *Combinatorial Pattern Matching*, pages 181–192, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

16 Juha Kärkkäinen and Peter Sanders. Simple Linear Work Suffix Array Construction. In *Proceedings of the 30th International Conference on Automata, Languages and Programming*, ICALP'03, pages 943–955, Berlin, Heidelberg, 2003. Springer-Verlag.

17 Toru Kasai, Gunho Lee, Hiroki Arimura, Setsuo Arikawa, and Kunsoo Park. Linear-Time Longest-Common-Prefix Computation in Suffix Arrays and Its Applications. In Amihood Amir, editor, *Combinatorial Pattern Matching*, pages 181–192, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg.

18 Donald E. Knuth. *The Art of Computer Programming, Volume 3: (2Nd Ed.) Sorting and Searching*. Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA, 1998.

19 U. Manber and G. Myers. Suffix Arrays: A New Method for On-Line String Searches. *SIAM Journal on Computing*, 22(5):935–948, 1993. doi:10.1137/0222058.

20 Giovanni Manzini. Two Space Saving Tricks for Linear Time LCP Array Computation. In Torben Hagerup and Jyrki Katajainen, editors, *Algorithm Theory - SWAT 2004*, pages 372–383, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

21 Ge Nong, Sen Zhang, and Wai Hong Chan. Linear Suffix Array Construction by Almost Pure Induced-Sorting. In James A. Storer and Michael W. Marcellin, editors, *DCC*, pages 193–202. IEEE Computer Society, 2009.

22 Simon J. Puglisi, W. F. Smyth, and Andrew H. Turpin. A Taxonomy of Suffix Array Construction Algorithms. *ACM Comput. Surv.*, 39(2), July 2007. doi:10.1145/1242471.1242472.

23 Robert Sedgewick. *Algorithms in C*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1990.

24 Wojciech Szpankowski. *Average Case Analysis of Algorithms on Sequences*. John Wiley & Sons, Inc., New York, NY, USA, 2001.

25 Brigitte Vallée. Dynamical sources in information theory: Fundamental intervals and word prefixes. *Algorithmica*, 29(1):262–306, February 2001. doi:10.1007/BF02679622.

26 Brigitte Vallée. The Depoissonisation Quintet: Rice-Poisson-Mellin-Newton-Laplace. In *Proceedings of the AofA 2018 Conference*, volume LIPICS Dagstuhl 110, pages 35:1–35:20, June 2018. (long version: https://arxiv.org/pdf/1802.04988). doi:10.4230/LIPIcs.AofA.2018.35.

27 Brigitte Vallée, Julien Clément, James Allen Fill, and Philippe Flajolet. The Number of Symbol Comparisons in QuickSort and QuickSelect. In *Automata, Languages and Programming, 36th International Colloquium, ICALP 2009, Rhodes, Greece, July 5-12, 2009, Proceedings, Part I*, volume 5555 of *Lecture Notes in Computer Science*, pages 750–763, 2009. doi:10.1007/978-3-642-02927-1_62.

A Appendix

A.1 Proof of Proposition 8. Relation (1)

Proof. Consider a node with prefix w in the trie $T(L)$. We denote by $[A_w, B_w]$ the maximal interval $[A, B]$ for which $w = \Gamma[A, B]$, and the decomposition of $[A_w, B_w]$ into largest dichotomic intervals, of the form

$$\texttt{Dic}(w) := \{[b_1, e_1], [b_2, e_2], \dots [b_k, e_k]\}, \quad \text{with } b_1 = A_w,\ e_{j-1} = b_j \text{ for } j \in [2\,..\,k], e_k = B_w\,.$$

We denote by $\texttt{dic}(w)$ the cardinality of $\texttt{Dic}(w)$, namely the number of dichotomic intervals it contains (with the previous notations, one has $\texttt{Dic}(w) = k$) .
This decomposition is easily built in a bottom-up strategy. We begin with the decomposition of $[A_w, B_w]$ into dichotomic intervals of length one in $D(L)$. As soon as there exists in the decomposition two adjacent nodes in $D(L)$ with the same predecessor, we replace them by their common predecessor. The recursive procedure halts at depth ℓ in $D(L)$ with two possible cases: one node of depth ℓ, or two adjacent nodes of depth ℓ with a different predecessor. For each dichotomic interval $[b, e]$ in $D(L)$ (distinct of the root), denote the predecessor of $[b, e]$ by $\texttt{pred}(b, e)$. We now prove that the two conditions are equivalent

- the prefix w satisfies the inequality $\Gamma[\texttt{pred}(b, e)] \prec w \preceq \Gamma[b, e]$;
- the dichotomic interval $[b, e]$ belongs to the set $\texttt{Dic}(w)$;

Indeed, when $[b, e] \in \texttt{Dic}(w)$, then first $\Gamma[b, e] \succeq w$, and second the interval $[b, e]$ is a largest dichotomic interval, then its predecessor $\Gamma[\texttt{pred}(b, e)]$ satisfies the strict inequality $\Gamma[\texttt{pred}[b, e]] \prec w$.
Conversely, when the prefix w satisfies the inequality $\Gamma[\texttt{pred}(b, e)] \prec w \preceq \Gamma[b, e]$, the interval $[b, e]$ is a largest dichotomic interval for which $\Gamma[b, e] \succeq w$, and thus the inequality $\Gamma[\texttt{pred}(b, e)] \prec w \preceq \Gamma[b, e]$ holds.

There is thus a bijection between the two multi-sets

$$\{\texttt{Dic}(w) \mid w \text{ internal node of } T(L)\}, \quad \{w \mid \exists (b, e) \in D(L), \ \ \Gamma[\texttt{pred}(b, e)] \prec w \preceq \Gamma[b, e]\}.$$

As the cardinality of the last multi-set coincides with the number of (positive) comparisons that are needed to compute the $\gamma(L)$ array, this ends the proof. ◀

A.2 Proof of Proposition 8. Bound for $\texttt{dic}(w)$ given in Eq. (2)

This bound will be established thanks to the next lemmas 10, 12 and 13.

▶ **Lemma 10.** *Let $n > 0$ and D_n be a dichotomic tree of height $h = \lceil \log_2 n \rceil$. Each node $(b, e) \in D_n$ corresponds to a dichotomic interval $[b, e]$ of length $e - b$. At depth $0 \leq \ell \leq h$, all nodes in D_n correspond with intervals of length either $\lfloor \frac{n}{2^\ell} \rfloor$ or $\lfloor \frac{n}{2^\ell} \rfloor + 1$.*

Proof. The root interval $[0, n]$ has length n. If n is even, left and right intervals children have both length $n/2$ and, if n is odd, left and right intervals children have respectively length $\lfloor n/2 \rfloor$ and $\lfloor n/2 \rfloor + 1$. Hence the property of the lemma holds at depth $\ell = 1$. By recursion, suppose now that at a given depth ℓ and posing $N = \lfloor \frac{n}{2^\ell} \rfloor$, all intervals have length either N or $N+1$. We can check easily (for instance using binary expansion of n) that $\lfloor N/2 \rfloor = \lfloor n/2^{\ell+1} \rfloor$, and also $\lfloor (N+1)/2 \rfloor = \lfloor n/2^{\ell+1} \rfloor$ if N is even and $\lfloor (N+1)/2 \rfloor = \lfloor n/2^{\ell+1} \rfloor + 1$ if N is odd. Hence the property holds at depth $\ell + 1$. ◀

So at a given depth, dichotomic intervals can only take two lengths. Abusing notation, for a fixed dichotomic tree D_n, we note $\texttt{Dic}(i, f)$ the decomposition of the interval $[i, f]$ as a union of minimal cardinality of disjoint (apart from their extremities) dichotomic intervals. We also define $\texttt{dic}(i, f)$ as this minimal cardinality. We now want to upper bound the parameter $\texttt{dic}(i, f)$. We will use the following definition:

▶ **Definition 11.** *For a dichotomic tree D_n, an interval (i, f) is said to be left (resp. right) aligned if there exists a dichotomic interval (b, e) such that $i = b$ and $e \geq f$ (resp. $f = e$ and $b \leq i$).*

First we examine the case of an interval (i, f) which is aligned on the left.

▶ **Lemma 12.** *Let $n > 0$ and a dichotomic tree D_n. Let consider a pair (b, f) with $0 \leq b < f \leq n$ such that $[b, f]$ is left aligned on a dichotomic interval $[b, e]$. We have*

$$\texttt{dic}(b, f) \leq 1 + \log_2(f - b).$$

Proof. If $[b, f]$ is dichotomic, $\texttt{dic}(b, f) = 1$ and the lemma holds. We assume now that $[b, f]$ is not dichotomic. There exists $e < e'$ such that

$$[b, e] \subsetneqq [b, f] \subsetneqq [b, e'],$$

and $[b, e], [b, e']$ are dichotomic intervals with $(b, e') = \texttt{pred}(b, e)$ (meaning $\lfloor (b + e')/2 \rfloor = e$). Consequently $[e, e']$ is also a dichotomic interval of D_n of length $b - e + \epsilon$ (for some $\epsilon \in \{0, 1\}$, see Lemma 10). The first and largest interval of $\texttt{Dic}(b, f)$ is the dichotomic interval $[b, e]$. We can thus write $\texttt{dic}(b, f) = 1 + \texttt{dic}(e, f)$. Since $[e, f]$ is left aligned on the dichotomic interval $[e, e']$ we can iterate. Using the fact that in the next iteration we decompose an interval $[e, f]$ of length $f - e \leq f/2$, we get the bound $\texttt{dic}(b, f) \leq 1 + 2\log_2(f - b)$. ◀

The lemma can be adapted when the interval is right aligned. We can now examine the general case (i, f) with $0 \leq i < f \leq n$ as illustrated in Fig. 7.

▶ **Lemma 13.** *Let $n > 0$ and a dichotomic tree D_n. Let consider a pair (i, f) with $0 \leq i < f \leq n$. We have*

$$\texttt{dic}(i, f) \leq 2 + 2\log_2(f - i).$$

Proof. Let $[b, e]$ the dichotomic interval of maximal length such that $[i, f] \subset [b, e]$. Then posing $m = \lfloor (b + e)/2 \rfloor$, we decompose the interval $[i, f]$ according to this splitting point m into two intervals $[i, m] \cup [m, f]$. Those two intervals are respectively aligned on the right and the left of two dichotomic consecutive intervals $[b, m]$ and $[m, e]$ of length less than $f - i$. We can also check that $\texttt{Dic}(i, f) = \texttt{Dic}(i, e) + \texttt{Dic}(e, f)$. Finally we apply the previous lemma to prove the result. ◀

Now going back to usual notation of the paper, this yields the estimate of Eq. (2) in Proposition 8.

A.3 Exact expression for $\texttt{dic}(w)$ in the pure dichotomic case

In the case of a pure dichotomy, there is an exact expression for $\texttt{dic}(w)$, in terms of the parameter $|x|_1$ that denotes the number of '1' in the binary writing of the integer x. This makes more precise the estimate given in Lemma 12.

▶ **Lemma 14.** *In the pure dichotomic case, the following holds:*

- *Consider a pair (b, f) such that the interval $[b, f]$ is left aligned. Then $\texttt{dic}(b, f) = |f - b|_1$.*
- *Consider a pair (i, e) such that the interval $[i, e]$ is right aligned on a dichotomic interval $[b, e]$. Then $\texttt{dic}(i, e) = |i - b|_1$*

Proof. The proof is omitted. ◀

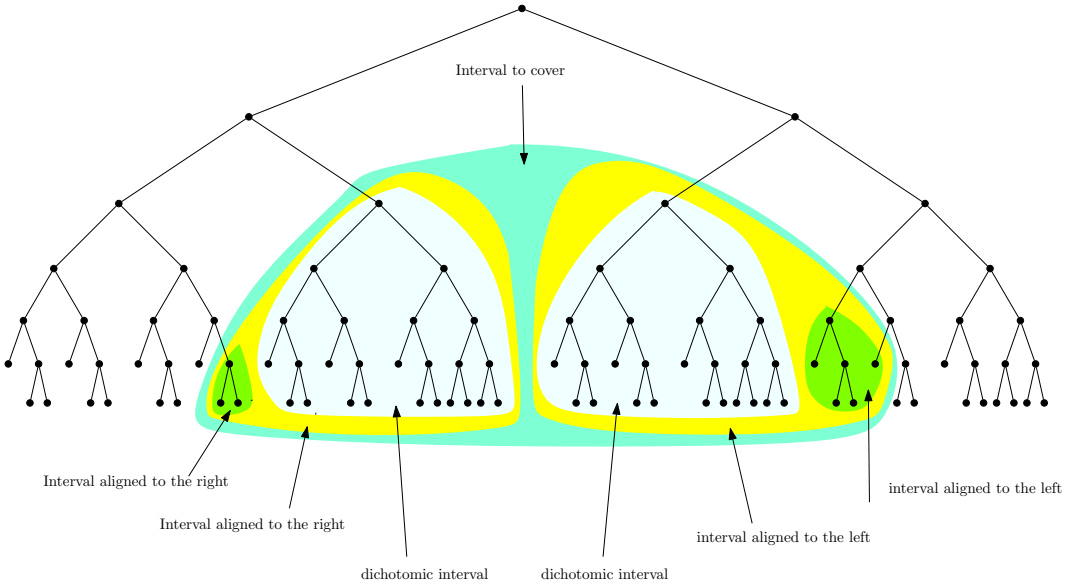


Figure 7 A dichotomic tree with an interval $\mathcal{I}$ (in light green). It decomposes in two intervals (yellow) which are respectively aligned on the left and on the right.

A.4 Proof of Theorem 9: Average-case analysis of a cost with toll in a trie built on a good general source

In this section, $x : \mathbb{N} \to \mathbb{R}$ denotes a toll function, and, for a trie built on a list of words L, X is the total cost defined in (5)

$$X(L) := \sum_{w|N_w \geq 2} x(N_w)\,,$$

where N_w is the number of elements of L which begin with w.

There are three main steps in the analysis of the expectation of parameter X.

First step. We begin to deal with the Poisson model $\mathcal{P}_z$, where the cardinality of the list N is a random variable which follows the law

$$\Pr[N = n] = e^{-z}\frac{z^n}{n!}\,.$$

With a toll x, we associate its Poisson transform

$$P_x(z) := e^{-z}\sum_{n\geq 0} x(n)\frac{z^n}{n!}\,,$$

which coincides with the expectation $\mathbb{E}_z[x]$ of the cost x in the Poisson model. In the model $\mathcal{P}_z$, the cardinality N_w follows a Poisson law of rate $z\,p_w$ that involves the fundamental probability p_w of the source. This is the main advantage of the Poisson model.

We then deal with the Poisson transforms $P_X(z)$ and $P_x(z)$ that resp. coincide with the expectations of X and x in the Poisson model, and averaging Relation (5) in the model $\mathcal{P}_z$ entails a relation between the two Poisson transforms

$$P_X(z) := \mathbb{E}_z[X] = \sum_{w\in\Sigma^\star} \mathbb{E}_z[x(N_w)] = \sum_{w\in\Sigma^\star} P_x(z\,p_w). \tag{7}$$

Then, the function $P_X(z)$ writes as a harmonic sum built on the function P_x. The Mellin transform is a good tool for dealing with harmonic sums and the Mellin transform $P_X^*(s)$ factorizes and involves the Λ generating function of the source (defined in (3)),

$$P_X^*(s) = \Lambda(-s) \cdot P_x^*(s)\,.$$

On the other side, the Mellin transform of P_x satisfies,

$$P_x^*(s) = \sum_{k\geq 2} \frac{x(k)}{k!} \int_0^\infty e^{-z} z^k z^{s-1} dz = \sum_{k\geq 2} \frac{x(k)}{k!} \Gamma(k+s) = \sum_{k\geq 2} \frac{x(k)}{k} \frac{\Gamma(k+s)}{\Gamma(k)}. \tag{8}$$

Second step. We now wish to return first to $P_X(z)$ (the expectation of X in the Poisson model), then extract the coefficients of order n in P_X to obtain the expectation $E_{[n]}[X]$ of X in the usual model where N is fixed and equal to n (this is called the Bernoulli model). This step is called the Depoissonisation step and may be performed with Depoissonisation techniques or via the Rice Formula, as it is explained in [26]. In [26], the author shows that the Rice method may be applied as soon as the function

$$\psi(s) := \frac{P_X^*(-s)}{\Gamma(-s)} = \frac{P_x^*(-s)}{\Gamma(-s)} \cdot \Lambda(-s)$$

is *tame* on a domain $\mathcal{R}$ (meaning "meromorphic on $\mathcal{R}$ and of polynomial growth when $|\Im s| \to \infty$ inside $\mathcal{R}$).

For the two costs that bound the function `Dic`, namely, the function $x(k) = 1$ (associated with the size of the trie) but also the function $x(k) = \log k$, the ratio $P_x^*(s)/\Gamma(s)$ is tame on a domain $\mathcal{R} := \{s \mid \Re s > 1 - a\}$ for some $a > 0$ and even analytic (without poles). In our context, the Λ function of the source is also tame on such a domain[7] $\mathcal{R}$.
The Rice method deals with the product of the two functions

$$\psi(s) := \frac{P_X^*(-s)}{\Gamma(-s)} = \frac{P_x^*(-s)}{\Gamma(-s)} \cdot \Lambda(-s), \qquad L_n(s) := \frac{(-1)^{n+1}\, n!}{s(s-1)(s-2)\dots(s-n)}$$

and provides the estimate for the expectation $E_{[n]}[X]$, as

$$E_{[n]}[X] = -\left[\sum_{k \mid s_k \in \mathcal{R}} \operatorname{Res}\left[L_n(s) \cdot \psi(s); s = s_k\right] + \frac{1}{2i\pi} \int_{\mathcal{C}} L_n(s) \cdot \psi(s)\, ds \right],$$

where the sum is taken over the poles s_k of ψ inside a region $\mathcal{R}$ of tameness. We have now to choose a region $\mathcal{R}$ for tameness, exhibit the poles that appear in this region and compute the associated residues.

Periodicity of the source. In this general context of good sources, the series $\Lambda(s)$ has always a simple pole at $s = 1$ inside the domain $\mathcal{R}$. And there are two possibilities

(*i*) The pole $s = 1$ is the unique pole on the vertical line $\Re s = 1$, with a residue equal to the inverse of the entropy, $1/h(\mathcal{M})$;

(*ii*) There exists another pole $s \neq 1$ on the line $\Re s = 1$, and then there is an infinite family of poles regularly spaced on this line, of the form $s_k = 1 + 2ik\rho\pi$ for some ρ and any $k \in \mathbb{Z}$.

[7] We say here that the source is good.

In the first case, the source is aperiodic; in the second case, the function $s \mapsto \Lambda(s)$ is periodic, and the source itself is said to be periodic. At a first glance, the case (ii) seems to be quite particular. However, as we explain in Section 4.1, this cases arises as soon as the source is *regular*, i.e. associated with an alphabet of size b, and probabilities $p_i = 1/b$. In this case, one has $\Lambda(s) = (1 - b^{1-s})^{-1}$ and the parameter ρ equals $1/(\log b)$.

The residue sum involves the function $P_x^\star$ defined in (8), and there are two possibilities for the residue sum:

$$\frac{n}{h(\mathcal{M})} \cdot P_x^*(1) \quad (\text{case } (i)), \qquad \frac{n}{h(\mathcal{M})} \left[\sum_{k \in \mathbb{Z}} n^{s_k - 1} \cdot P_x^*(s_k) \right] \quad (\text{case } (ii)).$$

Remark that the factor of the second sum provides a periodic function of the variable $\log_\rho n$. This ends the proof.

A.5 Towards the analysis of coincidence $\gamma(b, e)$

Consider a node (b, e) of depth ℓ in $D(L)$, the longest common prefix $\Gamma[b, e] := \Gamma[L_b, L_e]$, its length denoted as $\gamma(b, e)$, and the two parameters t_b, t_e of L_b, L_e. With any integer k, we associate the set $\mathcal{V}_k$ that gathers all the ends of fundamental intervals of the source $\mathcal{M}$ of depth k. There are three cases for the cardinality $J_k(b, e)$ of the intersection $\mathcal{V}_k \cap [t_b, t_e]$, i.e., $J_k(b, e) = 0, 1$ or $J_k(b, e) \geq 2$.

We first consider the toy case where : (a) the intervals $[t_b, t_e]$ exactly coincide with the binary intervals $[v_b, v_e]$ defined in Eq. (6) – (b) the source $\mathcal{M}$ is b-regular. Then, for each possible cardinality $J_k(b, e)$, we can relate first $\gamma(b, e)$ and k, then k and ℓ, and we obtain the estimate

$$\theta\ell - 1 \leq \gamma(b, e) \leq \theta(\ell + 2) \qquad \text{with} \quad \theta := \log 2/(\log b).$$

Of course, the toy case described in items (a) and (b) is not the actual situation; but the actual situation – *at least on average* and for $\ell \to \infty$– appears to be close to this "simplification", due to the two following facts:

(a') The interval $[t_b, t_e]$ is close to the interval $[v_b, v_e]$. For an interval (b, e) at depth ℓ, the pair (t_b, t_e) indeed admits the joint distribution $f_{b,e,n}$ on the triangle $\mathcal{T} := \{(x, y) \mid 0 \leq x \leq y \leq 1\}$ defined as

$$f_{b,e,n}(x, y) = \frac{n!}{(b-1)!(e-b-1)!(n-e)!} x^{b-1}(y-x)^{e-b-1}(1-y)^{n-e} \qquad (n = 2^p)\,.$$

It is then easy to exactly compute the expectation and the variance of the main parameters $t_b, t_e, |t_b - t_e|, (1/2)(t_b + t_e)$ which allows to compare the two intervals $[t_b, t_e]$ and $[v_b, v_e]$. These variances are always negligible with respect to their expectations (for $n = 2^p \to \infty$), and there is a *concentration* phenomenon for these distributions.

(b') For a good source, there is an asymptotic Gaussian law on the logarithms $\ell_k(\cdot)$ of the lengths of the fundamental intervals $I_k(\cdot)$ (for $k \to \infty$) (see [25]).