# From Nerode's congruence to Suffix Automata with mismatches

Maxime Crochemore, Chiara Epifanio, Alessandra Gabriele, Filippo Mignosi

# From Nerode's congruence to Suffix Automata with mismatches ⋆

M. Crochemore [a], C. Epifanio [b], A. Gabriele [b], F. Mignosi [c].

[a] *King's College London, UK*
*and Institut Gaspard-Monge, Université Paris-Est, France*
[b] *Dipartimento di Matematica e Applicazioni, Università di Palermo, Italy*
[c] *Dipartimento di Informatica, Università dell'Aquila, Italy*

**Abstract**

In this paper we focus on the minimal deterministic finite automaton $S_k$ that recognizes the set of suffixes of a word $w$ up to $k$ errors. As first results we give a characterization of the Nerode's right-invariant congruence that is associated with $S_k$. This result generalizes the classical characterization described in [5]. As second result we present an algorithm that makes use of $S_k$ to accept in an efficient way the language of all suffixes of $w$ up to $k$ errors in every window of size $r$ of a text, where $r$ is the repetition index of $w$. Moreover, we give some experimental results on some well-known words, like prefixes of Fibonacci and Thue-Morse words. Finally, we state a conjecture and an open problem on the size and the construction of the suffix automaton with mismatches.

*Key words:* Combinatorics on words, indexing, suffix automata, languages with mismatches, approximate string matching.

## 1 Introduction

One of the seminal results in string matching is that the size of the suffix automaton of a word, also called DAWG (directed acyclic word graph), is linear [4,5,11]. More precisely, in [4] it is proved that the size of the suffix

⋆ Partially supported by MIUR National Project PRIN "Automi e Linguaggi Formali: aspetti matematici e applicativi."

*Email addresses:* maxime.crochemore@kcl.ac.uk (M. Crochemore), epifanio@math.unipa.it (C. Epifanio), sandra@math.unipa.it (A. Gabriele), mignosi@di.univaq.it (F. Mignosi).

automaton is linear, while in [5] authors have given an on-line linear time algorithm to build this automaton. Minimality as an automaton accepting the suffixes is in [11].

These results are surprising as the maximal number of subwords that may occur in a word is quadratic according to the length of the word. Suffix trees are linear too, but they represent strings by pointers to the text, while DAWGs work without the need of accessing it.

Literature on languages with mismatches and corresponding data structures involves many results, among the most recent ones [2,6,8,9,14,15,17,20,22,25]. Most of these papers deal with approximate string matching and indexing. In particular, [22] presents the most efficient algorithm for approximate string matching with mismatches without indexing. Moreover, in [14,15,17] authors have considered some data structures recognizing words occurring in a text $w$ up to $k$ errors in each substring of length $r$ of the text. The presence of a window in which a fixed number of errors is allowed generalizes the classical $k$-mismatch problem and, at the same time, it allows more errors on the overall.

In this paper we focus on the minimal deterministic finite automaton that recognizes the set of suffixes of a word $w$ up to $k$ errors, denoted by $S_{w,k}$, or simply by $S_k$ if there are no risk of misunderstanding on $w$. Our contribution is mainly theoretical in nature, even if we give some experimental results that support a conjecture on the size of $S_k$. In the first part of the paper we present two main results. The first one is a characterization of the Nerode's right-invariant congruence that is associated with $S_k$. This results generalizes the classical characterization described in [5] (see also [13,21]). This classical characterization has been used to design an efficient construction of the suffix automaton with no mismatches, which, up to the set of final states, is also called DAWG. Up to now we have only extended Nerode's congruence to the approximate case and we have proved some theoretical results on it. We think these results will be useful for designing an efficient on-line algorithm for the construction of the suffix automaton with mismatches. It still remains an open problem how to design such an algorithm.

The second main result concerns the description of an algorithm that makes use of the automaton $S_k$ in order to accept, in an efficient way, the language of all suffixes of $w$ up to $k$ errors in every windows of size $\hat{r}$, for a specific integer $\hat{r}$ called the repetition index. To do this we first provide a linear-time algorithm that finds $\hat{r}$.

In the second part of the paper we show some empirical results. We have constructed the suffix automaton with mismatches of a great number of words and have considered overall its structure when the input word is well-known, such as the prefixes of Fibonacci and Thue-Morse words, as well as words

of the form $bba^n$, $a, b \in \Sigma, n \geq 1$, and some random words generated by memoryless sources. We have studied how the number of states grows w.r.t the length of the input word. Following our experiments on these classes of words, we conjecture that the (compact) suffix automaton with $k$ mismatches of any text $w$ has size $O(|w| \cdot \log^k(|w|))$.

Given a word $v$, Gad Landau wondered if there exists a data structure having a size "close" to $|v|$ that allows approximate pattern matching in time proportional to the query plus the number of occurrences. This question is still open, even if recent results are getting closer to a positive answer. If our conjecture turns out to be true, it would settle Landau's question as discussed at the end of this paper. Moreover, in this case our data structure may be used in some classical applications of approximate indexing and string matching, such as recovering the original signals after their transmission over noisy channels, finding DNA subsequences after possible mutations, text searching in case there are typing or spelling errors, retrieving musical passages, A.I. techniques in feature vector, and so on. It may be important even in other applications, like in the field of Web search tools when we deal with *agglutinative languages*, i.e. languages that mainly resort to suffixes and declinations such as many Finno-Uralic languages (like Hungarian, Finnish, and Estonian), or in the case of real-time proposal of alternative internet URL in Domain Name Servers, or for deeper analysis of biological sequences.

The remainder of this paper is organized as follows. In the second section we give some basic definitions. In the third section we describe a characterization of the Nerode's right-invariant congruence relative to $S_k$. The fourth section is devoted to describe an algorithm that makes use of the automaton $S_k$ to accept in an efficient way the language of all suffixes of $w$ up to $k$ errors in every window of size $\hat{r}$ of a text, where $\hat{r}$ is the value of the repetition index of $w$. The fifth section contains our conclusions and some conjectures on the size of the suffix automaton with mismatches based on our experimental results.

## 2 Basic definitions

Let $\Sigma$ be a finite set of symbols, usually called *alphabet*. A *word* or *string* $w$ is a finite sequence $w = a_1 a_2 \dots a_n$ of characters in the alphabet $\Sigma$, its length (i.e. the number of characters of the string) is defined to be $n$ and it is denoted by $|w|$. The set of words built on $\Sigma$ is denoted by $\Sigma^*$ and the empty word by $\epsilon$. We denote by $\Sigma^+$ the set $\Sigma^* \setminus \{\epsilon\}$. A word $u \in \Sigma^*$ is a *factor* (resp. a *prefix*, resp. a *suffix*) of a word $w$ if and only if there exist two words $x, y \in \Sigma^*$ such that $w = xuy$ (resp. $w = uy$, resp. $w = xu$). Notice that some authors call *substring* what we define as a factor. We denote by *Fact(w)* (resp. *Pref(w)*, resp. *Suff(w)*) the set of all factors (resp. prefixes, resp. suffixes) of a word $w$.

We denote an occurrence of a nonempty factor in a string $w = a_1a_2 \dots a_n$ at position $i$ ending at position $j$ by $w(i,j) = a_i \dots a_j$, $1 \leq i \leq j \leq n$. The length of the factor $w(i,j)$ is $j - i + 1$, and we say that $u$ *occurs* in $w$ at position $i$ if $u = w(i,j)$, with $|u| = j - i + 1$.

In order to handle languages with errors, we need a notion of distance between words. It is represented by the function $d : \Sigma^* \times \Sigma^* \to \mathbb{R}^+ \cup \{0\}$ defined between two strings $x$ and $y$ on the alphabet $\Sigma$ as the minimal cost of a sequence of operations that transform $x$ into $y$ (and $\infty$ if no such sequence exists). In most applications the possible operations are insertion, deletion, substitution and transposition. These operations lead to the definitions of different distances between words. In this work we consider the *Hamming distance* [27] that allows only substitutions, which cost 1 in the simplified definition. It is finite whenever $|x| = |y|$. In this case the inequalities $0 \leq d(x,y) \leq |x|$ hold.

In other words, given two strings $x$ and $y$ having the same length, the distance $d(x,y)$ between them is the minimal number of character substitutions that transform $x$ into $y$.

In the field of approximate string matching, typical approaches for finding a string in a text consist in considering a percentage $D$ of errors, or fixing the number $k$ of them. Instead, we use an hybrid approach introduced in [17] that considers a new parameter $r$ and allow at most $k$ errors for any factor of length $r$ of the text.

**Definition 1** *Let $w$ be a string over the alphabet $\Sigma$, and let $k, r$ be non negative integers such that $k \leq r$. A string $u \neq \epsilon$* occurs in $w$ at position $\ell$ up to $k$ errors in a window of size $r$ *or, simply,* $k_r$-occurs in $w$ at position $\ell$, *if one of the following two conditions holds:*

- $|u| < r \Rightarrow d(u, w(\ell, \ell + |u| - 1)) \leq k$;
- $|u| \geq r \Rightarrow \forall i, 1 \leq i \leq |u| - r + 1,\ \ d(u(i, i+r-1), w(\ell+i-1, \ell+i+r-2)) \leq k$.

*A string $u$ satisfying the above property has a* $k_r$-occurrence *of $w$. A string $u$ that $k_r$-occurs as a suffix of $w$ is a $k_r$-suffix of $w$.*

*The empty word $\epsilon$ $k_r$-occurs in $w$ at any position $\ell = 0, 1, \dots, |w|$. It also represents a $k_r$-suffix of $w$.*

We suppose that the text $w$ is non-empty, $r \geq 2$ and $0 \leq k \leq r$, otherwise the above definition would have no meaning. We denote by $L(w,k,r)$ (resp. $\mathit{Suff}(w,k,r)$) the set of words (resp. suffixes) $u$ that $k_r$-occur in $w$ at some position $\ell$, $1 \leq \ell \leq |w| - |u| + 1$. Notice that $L(w,k,r)$ is a *factorial language*, i.e. if $u \in L(w,k,r)$ then each factor of $u$ belongs to $L(w,k,r)$. Moreover, we denote by $\mathit{Suff}(w,k)$ the set of $k_r$-suffixes of $w$ for $r = |w|$.

**Remark 2** *The condition $r = |w|$ is equivalent to considering no window on $w$. Indeed, in this situation, the problem of finding all $k_r$-occurrences of a string $u$ in the text is equivalent to the $k$-mismatch problem, that consists in finding all occurrences of the string $u$ in $w$ with at most $k$ errors (cf. [19]).*

**Example 3** *Let $w = abaa$ be a string on the alphabet $\Sigma = \{a, b\}$. The set of words that $k_r$-occur in $w$, when $k = 1$ and $r = 2$, is*

$$L(w, 1, 2) = \{a, b, aa, ab, ba, bb, aaa, aab, aba, abb, baa, bab, bba, bbb, aaaa, aaab, abaa, abab, abba, bbaa, bbab, bbba\}.$$

*Notice that words $aab, aaab, bbab, bbba$ occur with one error every $r = 2$ symbols, but with two errors in the whole word. Hence, they belong to $L(w, 1, 2)$, but not to $L(w, 1, 4)$. Moreover,*

$$\text{Suff}(w, 1, 2) = \{a, b, aa, ab, ba, aaa, aab, baa, bab, bba, aaaa, aaab, abaa, abab, abba, bbaa, bbab, bbba\}$$

*and*

$$\text{Suff}(w, 1) = \{a, b, aa, ab, ba, aaa, baa, bab, bba, aaaa, abaa, abab, abba, bbaa\}.$$

Now we can recall the definition of the *repetition index*, denoted by $R(w, k, r)$, that plays an important role in the construction of an automaton recognizing the language $L(w, k, r)$ (cf. [17]).

**Definition 4** *The* repetition index *of a string $w$, denoted by $R(w, k, r)$, is the smallest integer $h$ for which all strings of this length $k_r$-occur at most once in the text $w$.*

The parameter $R(w, k, r)$ is well defined because the set of integers $h$ for which all strings of this length $k_r$-occur at most once in the text $w$ is always non empty since the integer $h = |w|$ satisfies the condition. Moreover, it is easy to prove that if $\frac{k}{r} \geq \frac{1}{2}$ then $R(w, k, r) = |w|$ (cf. [14]).

In [14] it is proved that $R(w, k, r)$ is a non-increasing function of $r$ and a non-decreasing function of $k$, and that the equation $r = R(w, k, r)$ admits an unique solution $\hat{r}$. Moreover, it is proved, under some hypothesis on the source, that $R(w, k, r)$ has an upper bound that is almost surely logarithmic in the size of the text $w$. Recall that a sequence of random variables $X_n$ converges to a random variable $X$ almost surely, denoted $X_n \to X$ (a.s.), if

$$\forall \epsilon > 0, \quad \lim_{N \to \infty} Pr\{\sup_{n \geq N} |X_n - X| < \epsilon\} = 1.$$

Almost surely convergence implies convergence in probability (see also [28,

Chapther 2] for further details).

**Remark 5** *In [17] authors gave an algorithm for building a deterministic finite automaton (DFA) recognizing the language $L(w,k,r)$ of all words that $k_r$-occur in the string $w$. They proved that the size of such automaton $\mathcal{A}(w,k,r)$ is bounded by a function that depends on the length $|w|$ of the text $w$, the repetition index $R(w,k,r)$, and the number $k$ of errors allowed in windows of size $\hat{r}$ unique solution of the equation $r = R(w,k,r)$, namely $|\mathcal{A}(w,k,r)| = O(|w| \cdot (\hat{r})^{k+1})$.*
*In the worst case, when both $\hat{r}$ and $k$ are proportional to $|w|$, the size of the automaton $\mathcal{A}(w,k,r)$ is exponential. But, under the hypothesis that $w$ is a sequence generated by a random memoryless source with identical symbol probabilities and that the number $k$ of errors is fixed for any window of size $\hat{r}$, the size of this automaton is*

$$O(|w| \cdot \log^{k+1}(|w|)) \text{ almost surely.}$$

Starting from the automaton $\mathcal{A}(w,k,r)$, an automaton recognizing the language *Suff*$(w,k,r)$ can be simply deduced from a procedure that first builds the automaton $\mathcal{A}(w\$,k,r)$, extending the alphabet $\Sigma$ by letter \$, then sets as terminal states only those states from which an edge labeled by letter \$ outgoes, and finally removes all edges labeled \$ and the state they reach [13]. In this paper we focus on the minimal automaton recognizing *Suff*$(w,k)$. It is therefore natural to study the Nerode's congruence corresponding to it.

## 3 On the Nerode's congruence

In this section, we introduce a right-invariant congruence relation on $\Sigma^*$ used to define the suffix automaton of a word up to mismatches and we prove some properties of it. In particular we give a characterization of the Nerode's congruence relative to $S_k$. This result generalizes a classical result described in [5] (see also [13,21]), where it is used in an efficient construction of the suffix automaton with no mismatches, that is also called DAWG (directed acyclic word graph), up to the set of final states. We think that it is possible to define such an algorithm even when dealing with mismatches. It would be probably more complex than the classical one.

In what follows, we do not consider the window, i. e. we set $r = |w|$.

Let us start by introducing the following definition, that is a generalization of the one given in [5].

**Definition 6** *Let $w = a_1 \dots a_n$ be a word in $\Sigma^*$. For any word $y \in \Sigma^*$, the end-set of $y$ in $w$ up to $k$ mismatches, denoted by end-set$_w(y,k)$, is the set*

*of all final positions in which $y$ $k$-occurs in $w$, i.e. end-set$_w(y,k) = \{j \mid y\, k$-occurs in $w$ with ending position $j\}$. Notice that end-set$_w(\epsilon,k) = \{0,1,\ldots,n\}$.*

By using Definition 6 we can define an equivalence relation between words on $\Sigma^*$.

**Definition 7** *Two words $x$ and $y$ in $\Sigma^*$ are end$_k$-equivalent, or $\equiv_{w,k}$, on $w$ if the following two conditions hold.*

*(1) end-set$_w(x,k) =$ end-set$_w(y,k)$;*
*(2) for any position $i \in$ end-set$_w(x,k) =$ end-set$_w(y,k)$, the number of errors available in the suffix of $w$ starting at position $i+1$ is the same after the reading of $x$ and of $y$, i.e.*
$$\min\{|w|-i, k-err_i(x)\} = \min\{|w|-i, k-err_i(y)\},$$
*where $err_i(u)$ is the number of mismatches of the word $u$ that $k_r$-occurs in $w$ with final position $i$.*

*We denote by $[x]_{w,k}$ the equivalence class of $x$ with respect to $\equiv_{w,k}$. The degenerate class is the equivalence class of words that have no $k$-occurrences in $w$ (i.e., words with empty end-set in $w$ up to $k$ mismatches).*

In other words, two words $x$ and $y$ in $\Sigma^*$ are *end$_k$*-equivalent if, besides having the same end-set in $w$ up to $k$ mismatches as in the exact case [5], the number of errors available in the suffix of $w$ after the reading of $x$ and of $y$ is the same. The definition includes two cases depending on the considered final position $i \in$ *end-set$_w(x,k) =$ end-set$_w(y,k)$* of $x$ and $y$ in $w$:

2.a) if this position is sufficiently "far from" the end of the word, which means that $|w| - i \geq \max\{k-err_i(x), k-err_i(y)\}$, then the number of errors available after this position is the same in both cases, i.e. $k - err_i(x) = k - err_i(y)$, which implies that $err_i(x) = err_i(y)$. In this case
$$\min\{|w|-i, k-err_i(x)\} = k - err_i(x) = k - err_i(y) = \min\{|w|-i, k-err_i(y)\}.$$
2.b) otherwise, if this position is sufficiently "near" the end of the word, which means that $|w| - i \leq \min\{k-err_i(x), k-err_i(y)\}$, then mismatches are possible in any position of the suffix of $w$ having length $|w|-i$. This does not necessarily imply that $err_i(x) = err_i(y)$. Therefore
$$\min\{|w|-i, k-err_i(x)\} = |w|-i = \min\{|w|-i, k-err_i(y)\}.$$

**Example 8** *Let us consider the prefix of length* 10 *of the Fibonacci word, $w = abaababaab$, and let us suppose that the number $k$ of errors allowed in any factor is* 2.

- *The words $x = baba$ and $y = babb$ have the same* end-set*, that is* end-set$_w(baba, 2) = \{5,6,8,10\} =$ end-set$_w(babb, 2)$*, but the two words are not*

*$end_k$-equivalent because it is not true that for any position* $i \in$ *end-set*$_w(baba, 2)$ $=$ *end-set*$_w(babb, 2)$*, the number of errors available in the suffix of* $w$ *having* $i+1$ *as first position is the same after the reading of* $x$ *and of* $y$*. In fact, if we consider* $i = 5$*,*

$$err_5(baba) = 2 \text{ and } err_5(babb) = 1$$

*and then*

$$\min\{|w| - 5, 2 - err_5(baba)\} = 0 \neq 1 = \min\{|w| - 5, 2 - err_5(babb)\}.$$

- *The words* $x = abaababa$ *and* $y = baababa$ *are trivially* $end_k$*-equivalent because they have the same* end-set*, that is* end-set$_w(abaababa, 2) = \{8\} =$ end-set$_w(baababa, 2)$*, and for* $i = 8$ *the number of errors available in the suffix of* $w$ *having* $i+1$ *as first position is the same after the reading of* $x$ *and of* $y$*. In fact, if we consider* $i = 8$*,*

$$err_8(abaababa) = 0 \text{ and } err_8(baababa) = 0$$

*and then*

$$\min\{|w| - 8, 2 - err_8(abaababa)\} = 2 = \min\{|w| - 8, 2 - err_8(baababa)\}.$$

- *The words* $x = abaababaa$ *and* $y = baababab$ *have the same* end set*, that is* end-set$_w(abaababaa, 2) = \{9\} =$ end-set$_w(baababab, 2)$*, and for* $i = 9$ *the number of errors available in the suffix of* $w$ *having* $i+1$ *as first position is the same after the reading of* $x$ *and of* $y$*, even if*

$$err_9(abaababaa) = 0 \text{ and } err_9(baababab) = 1.$$

*In fact, one has that*

$\min\{|w| - 9, 2 - err_9(abaababaa)\} = 1 = \min\{|w| - 9, 2 - err_9(baababab)\}$*, and then* $x$ *and* $y$ *are* $end_k$*-equivalent.*

The following lemma and theorem summarize some properties of $end_k$-equivalence. Before stating them, we recall that an equivalence relation $\equiv$ on $\Sigma^*$ is *right invariant* if, for any $x, y, z \in \Sigma^*$, $x \equiv y$ implies that $xz \equiv yz$.

**Lemma 9***(i)* $\equiv_{w,k}$ *is a right-invariant equivalence relation on* $\Sigma^*$*.*

*(ii) If* $x$ *and* $y$ *are* $end_k$*-equivalent on* $w$ *and* end-set$_w(x, k) =$ end-set$_w(y, k) \neq \emptyset$*, then one is a suffix of the other up to* $2k$ *errors.*

*(iii) Words* $xy$ *and* $y$ *are* $end_k$*-equivalent on* $w$ *if and only if for any* $i \in$ *end-set*$_w(xy, k) =$ *end-set*$_w(y, k)$*, the* $k$*-occurrence of* $y$ *with ending position* $i$ *is immediately preceded by a* $t$*-occurrence of* $x$*, where* $t = \max\{(k - err_i(y)) - (|w| - i), 0)\}$*.*

**PROOF.**

(i) $\equiv_{w,k}$ is an equivalence relation. Indeed it is obviously reflexive, symmetric and transitive.

Moreover this relation is a right-invariant equivalence. For any $x, y \in \Sigma^*$, if $x \equiv_{w,k} y$, then *end-set*$_w(x, k) =$ *end-set*$_w(y, k)$ and for any position $i \in$ *end-set*$_w(x, k) =$ *end-set*$_w(y, k)$, $\min\{|w| - i, k - err_i(x)\} = \min\{|w| - i, k -$

$err_i(y)\}$. Since the number of errors available in the suffix of $w$ having $i$ as first position is the same after the reading of $x$ and of $y$, then for any $z \in \Sigma^*$ $xz$ is a $k$-occurrence of $w$ if and only if $yz$ is a $k$-occurrence of $w$. Hence, $\textit{end-set}_w(xz,k) = \textit{end-set}_w(yz,k)$ and for any position $j \in \textit{end-set}_w(xz,k) = \textit{end-set}_w(yz,k)$ the number of errors available in the suffix of $w$ having $j$ as first position is the same after the reading of $xz$ and of $yz$.

(ii) By definition $x$ and $y$ are such that $\textit{end-set}_w(x,k) = \textit{end-set}_w(y,k)$. Therefore, for any $i \in \textit{end-set}_w(x,k) = \textit{end-set}_w(y,k)$, both $x$ and $y$ $k$-occur in $w$ with final position $i$, which implies that one of them is a suffix of the other up to mismatches. By the triangular inequality, $d(x,y) \leq d(x,w)+d(w,y) \leq 2k$. Hence, by definition, $x$ and $y$ are one a $2k$-suffix of the other.

(iii) Let us suppose, by hypothesis, that $xy \equiv_{w,k} y$. Therefore, $\textit{end-set}_w(xy,k) = \textit{end-set}_w(y,k)$ and for any position $i \in \textit{end-set}_w(xy,k) = \textit{end-set}_w(y,k)$, $\min\{|w|-i, k-err_i(xy)\} = \min\{|w|-i, k-err_i(y)\}$. Since $err_i(y) \leq err_i(xy)$, then $k-err_i(xy) \leq k-err_i(y)$. For any $i \in \textit{end-set}_w(xy,k) = \textit{end-set}_w(y,k)$, we can distinguish the following cases.

(1) Let $\min\{|w|-i, k-err_i(y)\} = k-err_i(y)$. Since $k-err_i(xy) \leq k-err_i(y) \leq |w|-i$, then $\min\{|w|-i, k-err_i(xy)\} = k-err_i(xy)$. Since $\min\{|w|-i, k-err_i(xy)\} = \min\{|w|-i, k-err_i(y)\}$, then $k-err_i(xy) = k-err_i(y)$ and all the $err_i(xy)$ errors are in $y$ and $x$ occurs exactly in $w$.

(2) Let $\min\{|w|-i, k-err_i(y)\} = |w|-i$. Since $\min\{|w|-i, k-err_i(xy)\} = \min\{|w|-i, k-err_i(y)\}$, one has that the number of errors available in the suffix of $w$ having $i+1$ as first position is $\min\{|w|-i, k-err_i(xy)\} = |w|-i$. Therefore the maximal allowed number of errors in $x$ is $k-[err_i(y)+(|w|-i)] \geq 0$.

Hence, for any $i \in \textit{end-set}_w(xy,k) = \textit{end-set}_w(y,k)$, the $k$-occurrence of $y$ with final position $i$ is immediately preceded by a $t$-occurrence of $x$, where $t = \max\{(k-err_i(y)-(|w|-i), 0)\}$.

Let us suppose, now, that for any $i \in \textit{end-set}_w(xy,k) = \textit{end-set}_w(y,k)$, the $k$-occurrence of $y$ with final position $i$ is immediately preceded by a $t$-occurrence of $x$, where $t = \max\{(k-err_i(y))-(|w|-i), 0)\}$. By hypothesis, $\textit{end-set}_w(xy,k) = \textit{end-set}_w(y,k)$. Let us distinguish two cases.

(1) Let us consider positions $i \in \textit{end-set}_w(xy,k) = \textit{end-set}_w(y,k)$ such that $t = 0$. In this case all the $err_i(xy)$ are in $y$ and the number of errors available in the suffix of $w$ having $i+1$ as first position is the same after the reading of $xy$ and of $y$.

(2) Let us, now, consider positions $i \in \textit{end-set}_w(xy,k) = \textit{end-set}_w(y,k)$ such that $t = (k-err_i(y))-(|w|-i)$. In this case $k-err_i(y) \geq |w|-i$ and then $\min\{|w|-i, k-err_i(y)\} = |w|-i$. By hypothesis, this $k$-occurrence of $y$ is immediately preceded by an occurrence of $x$ up to $t = (k-err_i(y))-(|w|-i)$ errors. Therefore, $k-err_i(xy) \geq k-[k-(err_i(y)+|w|-i)+err_i(y)] = |w|-i$ and $\min\{|w|-i, k-err_i(xy)\} = |w|-i$. □

**Theorem 10** *Words $x$ and $y$ are $end_k$-equivalent if and only if they have the*

*same future in $w$, i.e. for any $z \in \Sigma^*$, $xz$ is a $k$-suffix of $w$ if and only if $yz$ is a $k$-suffix of $w$.*

**PROOF.** By Lemma 9($i$), if $x$ and $y$ are $end_k$-equivalent, then for any $z \in \Sigma^*$ $xz$ and $yz$ are $end_k$-equivalent and then $xz$ is a $k$-suffix if and only if $yz$ is a $k$-suffix.

Let us suppose, now, that for any $z \in \Sigma^*$, $xz$ is a $k$-suffix if and only if $yz$ is a $k$-suffix. Therefore $\textit{end-set}_w(x,k) = \textit{end-set}_w(y,k)$. Moreover, for any $z$ such that $xz$ and $yz$ are suffixes of $w$, the ending position $i$ of $x$ and $y$ is such that $|w| - i = |z|$. By hypothesis, we can have two cases depending on $|z|$.

- Suppose that $z$ is such that $|z| \leq \min\{k - err_i(x), k - err_i(y)\}$. For such $z$ one has that $\min\{|w|-i, k-err_i(x)\} = |w|-i$ and $\min\{|w|-i, k-err_i(y)\} = |w| - i$ and the thesis is proved.
- Suppose that $z$ is such that $|z| \geq \max\{k - err_i(x), k - err_i(y)\}$. For such $z$ one has that $\min\{|w| - i, k - err_i(x)\} = k - err_i(x)$ and $\min\{|w| - i, k - err_i(y)\} = k - err_i(y)$. By hypothesis, for any position $i \in \textit{end-set}_w(x,k) = \textit{end-set}_w(y,k)$, any word $z \in \Sigma^*$ having $i+1$ as first position is such that $xz$ is a $k$-suffix of $w$ if and only if $yz$ is a $k$-suffix of $w$ and then $k - err_i(x) = k - err_i(v)$ and the thesis is proved. □

In what follows we use the term *partial* DFA (with respect to the alphabet $\Sigma$) for a deterministic finite automaton in which each state has not necessarily a transition for every letter of $\Sigma$. The smallest partial DFA for a given language is the partial DFA that recognizes the language and has the smallest number of states. It is called the minimal DFA recognizing the language. Uniqueness follows from Nerode's Theorem [23] of the right invariant equivalence relation.

By using Nerode's theorem and by Theorem 10 we have the following result.

**Corollary 11** *For any $w \in \Sigma^*$, the (partial) deterministic finite automaton having input alphabet $\Sigma$, state set $\{[x]_{w,k} \mid x \text{ is a } k \text{ occurrence of } w\}$, initial state $[\epsilon]_{w,k}$, accepting states those equivalence classes that include the $k$-suffixes of $w$ (i.e., whose end-sets include the position $|w|$) and transitions $\{[x]_{w,k} \xrightarrow{a} [xa]_{w,k} \mid x \text{ and } xa \text{ are } k\text{-occurrences of } w\}$, is the minimal deterministic finite automaton, denoted by $\mathrm{S}_{w,k}$ (or simply by $\mathrm{S}_k$ if there are no risks of misunderstanding on $w$), which recognizes the set Suff$(w,k)$.*

**PROOF.** Since the union of the equivalence classes that form the accepting states of $S_k$ is exactly the set of all $k$-suffixes of $w$, by Nerode's Theorem one

has that $S_k$ recognizes the language of all $k$-suffixes of $w$, i.e. the set $\mathit{Suff}(w,k)$. The minimality follows by Theorem 10. □

**Remark 12** *We note that* Suff(w,k)=Suff(w,k,r) *with* $r = |w|$*, which is equivalent to saying that there are at most* $k$ *errors in the entire word without window, and that* Suff(w,k,r) $\subseteq L(w,k,r)$*.*

## 4 Allowing more mismatches

In this section we present the second main result of the paper, that is the description of an algorithm that makes use of the automaton $S_k$ in order to accept, in an efficient way, the language $\mathit{Suff}(w,k,\hat{r})$ of all suffixes of $w$ up to $k$ errors in every window of size $\hat{r}$, that is the unique solution of the equation $r = R(w,k,r)$. As proved in [14–16] from $\hat{r}$ up to $|w|$ the repetition index gets constant, i.e. for any value $t \geq \hat{r}$ one has that $\hat{r} = R(w,k,t)$. This is also valid when the parameter $t$ is such that $t = |w|$, which implies that $\hat{r} = R(w,k,|w|) = R(w,k)$. These two extremal cases are the two cases we are considering. This fact implies that any word $u$ of length $|u| = \hat{r}$ has the following property: if $u$ $k_{\hat{r}}$*-occurs* or $k$*-occurs in* $w$, then, in both cases, it occurs only once in the text $|w|$.

Before describing our algorithm, we give a preliminary result that is important both for the following and on its own.

**Lemma 13** *Given the automaton* $\mathrm{S}_k$*, there exists a linear-time algorithm that returns* $\hat{r}$*.*

**PROOF.** In this proof we consider $w$ to be fixed. Let $q_0$ be the initial state of $S_k$ and let $\delta$ be its transition function. We call $\delta^*$ its extension to words, i.e. $\delta^*(q,w)$ is recursively defined as $\delta^*(q,w) = q$ if $w$ is the empty word and $\delta^*(q,w) = \delta(\delta^*(q,w^-),a)$, where $w^-$ is the word $w$ without the last letter and $a$ is the last letter of $w$.

If $u$ $k$-occurs twice in $w$, then from state $q' = \delta^*(q_0,u)$ there are two paths having different lengths to a final state. Conversely, if from a state $q'$ there are two paths having different lengths to a final state, then any word $u$ such that $q' = \delta^*(q_0,u)$ $k$-occurs twice in $w$. Therefore $\hat{r}-1$ is the greatest length of all words that reach a state from which there are two paths having different lengths to a final state. In what follows in this proof we will describe an algorithm that finds $\hat{r}-1$ in linear time and outputs $\hat{r}$.

We firstly find all states $q'$ such that there are two paths having different lengths from it to a final state. For "finding" these states we mean that we

will add a flag information to each state and we will turn this flag information to "red" only to these states.

The graph $G$ underlying $S_k$ is a directed acyclic graph ($DAG$ in short) because the language of $S_k$ is finite. Thus, the inverse $\hat{G}$ of $G$, that is the graph where all arcs are inverted, is also a $DAG$. In order to simplify the algorithm we add an initial state $\hat{q}$ to $\hat{G}$ that goes with one arc to each final state of $S_k$. It is well known that it is possible to find all single source minimal paths in linear time in a $DAG$ and an algorithm $\mathcal{V}$ is described in [10]. We refer to such algorithm $\mathcal{V}$ and to each step of it we add at most a constant number of extra steps. Roughly speaking, if during the execution of this algorithm the field distance of a state has been updated more than once, then the flag information has to be turned in red. More in details, the flag information can be white, green or red. The flag white means that the node has not yet been met during the execution of $\mathcal{V}$, the green one means that, up that moment during the execution of $\mathcal{V}$, the node has been encountered at least once and that the field distance has not been updated more than once i.e. all paths in $G$ to a final state have same lengths. The flag red means that there are at least two paths in $G$ to a final state having different lengths, or that, equivalently, the field distance has been updated more than once. The flag of the initial state $\hat{q}$ is set to be green and its distance is set to 0 while the flags of all other nodes are set to be white and distance equal to $+\infty$. If a node with white flag is reached starting from a node with a green flag, then its flag is set to green and its distance becomes the distance from the initial state, i.e. the distance of previous node plus one. If a node with green flag is reached starting from another node with a green flag and if its distance is equal to the distance of previous node plus one, then the flag remains green, otherwise it is set to red. Red flags propagate in the sense that the flag information of any state that is reached starting from a state with a red flag is set to be red. The fact that red flags propagate corresponds to the fact that if in $G$ there is an arc from $q'$ to $q''$ (i.e. there is an arc from $q''$ to $q'$ in $\hat{G}$) and from $q''$ there are two paths having different lengths to a final state then also from $q'$ there are two paths having different lengths to a final state.

The formal proof that the flag information is red only for all states $q'$ such that there are two paths having different lengths from it to a final state in $G$, is left to the reader.

At this point we have to find the greatest length of all words that reach a state having red flag in $S_k$ or, equivalently, we have to find the length of a *maximal* path in $G$ from $q_0$ to a state with a red flag. Since $G$ is a $DAG$, we now behave analogously as suggested in [10] as an application of the linear time algorithm $\mathcal{V}$ for single source shortest paths in $DAG$'s. The application described in [10] concerns PERT's diagrams where one has to find longest paths and not anymore shortest paths. A simple solution is to set the weight

of each arc in $G$ to be equal to $-1$ and look for the shortest path and the minimal distance. Changing the signs to these distance in each node will give the length of the longest path to each node.

The repetition index minus one $\hat{r} - 1$ is the greatest value among all nodes having red flag. The algorithm outputs $\hat{r}$. □

**Remark 14** *As a side effect of this algorithm, each state of the automaton* $\mathrm{S}_k$ *is equipped with an integer that represents a distance from this state to the end. For this purpose, it is sufficient to make a linear-time visit of the automaton.*

Now we can describe the algorithm that accepts the language $\mathit{Suff}(w, k, \hat{r})$. It makes use of $S_k$ and the value $\hat{r}$ computed by previous algorithm. We can distinguish two cases.

i) If a word $u$ has a length no more than $\hat{r}$ then we check if the word $u$ is accepted by the automaton $S_k$. If $u$ is accepted by this automaton then it is in the language $\mathit{Suff}(w, k, \hat{r})$, otherwise it does not belong to the language.
ii) If a word $u$ has a length greater than $\hat{r}$ then we consider its prefix $u'$ of length $\hat{r}$. Let $q$ be the state that is reached after reading $u'$ and let $i$ be the integer associated with this state (cf. Remark 14). We have that $|w| - i - \hat{r} + 1$ is the unique possible initial position of $u$. Given a position, checking whether a word $k_{\hat{r}}$-occurs at that position in the text $w$ can be done in linear time.

**Remark 15** *A simple variation of above algorithm allows us to find a pattern word in the text with more mismatches than* $k$ *in each window of size* $\hat{r}$*. It is sufficient that its prefix of length* $\hat{r}$ *occurs in the text up to* $k$ *mismatches, while the remaining suffix can have an arbitrary large number of mismatches. As stated in ii) of previous algorithm, this can be done in linear time.*

## 5 Experimental results and conclusions

This section is devoted to some experimental results and conjectures following from these.

We have constructed the suffix automaton with mismatches of a great number of words and we have considered overall its structure when the input word is well-known, such as the prefixes of Fibonacci and Thue-Morse words, as well as words of the form $bba^n$, $a, b \in \Sigma, n \geq 1$ and some random words. We have studied how the number of states grows w.r.t. the length of the input word. The algorithm we have used for the construction of the suffix automaton with

mismatches is neither efficient nor on-line. This is the reason for leaving out its description in this paper.

Concerning the prefixes of length $n$ of the Fibonacci word, in the non approximate case, a result in [7], together with a result in [5], implies that the following result holds.

**Proposition 16** *The suffix automaton of any prefix $v$ of the Fibonacci word $f$ has $|v|+1$ states.*

**PROOF.** In [7] it is proved that there exists an infinite set $X$ of prefixes of the Fibonacci word ($X$ is the set of the *special* prefixes) such that for any $v$ in $X$ the suffix automaton of $v$ has $|v|+1$ states. In [5] it is proved that the suffix automaton of $va$ has either 1 or 2 states more than the suffix automaton of $v$, for any $v$ in $\Sigma^*$, $a$ in $\Sigma$. Therefore, for any prefix $v$ of the Fibonacci word $f$, its suffix automaton has $|v|+1$ states. □

**Remark 17** *The result in [7] holds for the whole class of special Sturmian words. Therefore, above proposition can be extended to this class of infinite words.*

Our experimental results in the approximate case have led us to the following sequence $\{q'_n\}_n$, representing the number of states of the suffix automaton with one mismatch:

$$\{q'_n\}_n = 2, 4, 6, 11, 15, 18, 23, 28, 33, 36, 39, 45, 50, 56, 61, 64, 67, 70, 73, 79, 84, 90,$$
$$96, 102, 107, 110, 113, 116, 119, 122, 125, 128, 134, 139, 145, 151, 157, 163, 169, 175,$$
$$180, 183, 186, 189, 192, 195, 198, 201, 204, 207, 210, 213, 216, 222, 227, 233, 239, 245,$$
$$251, 257, 263, 269, \ldots$$

This means that the sequence of differences between two consecutive terms is:

$$\{q'_{n+1}-q'_n\}_n = 2, 2, 5, 4, 3, 5, 5, 5, 3, 3, 6, 5, 6, 5, 3, 3, 3, 3, 6, 5, 6, 6, 6, 5, 3, 3, 3, 3, 3,$$
$$3, 3, 6, 5, 6, 6, 6, 6, 6, 6, 5, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 6, 5, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6,$$
$$5, 3, 3, 3, 3, 3, 3, 3, 3, \ldots.$$

We note that after an initial part, there are $(fib_{i-1}-2)$ consecutive 6s, one 5, $(fib_i-1)$ consecutive 3s, one 6, one 5,etc, where $fib_i$ denotes the $i$-th Fibonacci number, $i = 4, 5, 6 \ldots$ . This leads to the following recursive formula:

$$q'_{fib_n} = q'_{fib_{n-1}} + 3(q'_{fib_{n-3}} - 1) + 10 + 6(q'_{fib_{n-4}} - 1).$$

From this recursion an explicit formula is easy to find. We did not prove the rule that describes the growth of the suffix automaton with one mismatch, but we checked that this rule holds true up to prefixes of length 2000 of the Fibonacci word $f$.

**Conjecture 18** *The size of the suffix automaton with one mismatch of the prefixes of the Fibonacci word grows according to the recursive formula* $q'_{fib_n} = q'_{fib_{n-1}} + 3(q'_{fib_{n-3}} - 1) + 10 + 6(q'_{fib_{n-4}} - 1)$.

Concerning the size of the suffix automaton with two mismatches of the prefixes of the Fibonacci word, our experimental results have led us to the following sequence $\{q''_n\}_n$, representing the number of states of the suffix automaton with two mismatches:

$$\{q''_n\}_n = 2, 3, 6, 10, 18, 27, 38, 49, 63, 73, 82, 103, 127, 148, 168, 182, 191, 200, 209,$$
$$230, 251, 276, 298, 320, 340, 354, 365, 374, 383, 392, 401, 410, 431, 452, 475, 501,$$
$$525, 547, 569, 591, 611, 625, 636, 647, 658, 667, 676, 685, 694, 703, 712, 721, 730,$$
$$751, 772, 795, 819, 843, 869, 893, 917, 939, 961, 983, 1005, 1027, 1047, 1061, 1020,$$
$$1072, 1083, 1094, 1105, 1116, 1127, 1136, 1145, 1154, 1163, 1172, 1181, 1190, 1199,$$
$$1208, 1217, 1226, 1235, 1244, 1265, 1286, 1309, 1333, 1357, 1381, 1405, 1429 \ldots$$

This means that the sequence of differences between two consecutive terms is:

$$\{q''_{n+1} - q''_n\}_n = 1, 3, 4, 8, 9, 11, 14, 10, 9, 21, 24, 21, 20, 14, 9, 9, 9, 21, 21, 25, 22, 22,$$
$$20, 14, 11, 9, 9, 9, 9, 9, 21, 21, 23, 26, 24, 22, 22, 22, 20, 14, 11, 11, 11, 9, 9, 9, 9, 9, 9,$$
$$9, 9, 21, 21, 23, 24, 24, 26, 24, 24, 22, 22, 22, 22, 22, 20, 14, 11, 11, 11, 11, 11, 11,$$
$$9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 21, 21, 23, 24, 24, 24, 24, 24, \ldots.$$

We note that after an initial part, there is one 23,$(fib_{i-2} - 3)$ consecutive 24s, one 26, $(fib_{i-3} - 1)$ consecutive 24s, $(fib_{i-2})$ consecutive 22s, one 20, one 14, $(fib_{i-1} - 2)$ consecutive 11s, $fib_i$ consecutive 9s, two consecutive 21s etc, where $fib_i$ denotes the $i$-th Fibonacci number, $i = 6, 7, \ldots$ .

This leads to the following recursive formula:

$$q''_{fib_n} = q''_{fib_{n-1}} + 125 + 24(q''_{fib_{n-5}} - 4) + 22(q''_{fib_{n-6}}) + 11(q''_{fib_{n-5}} - 2) + 9(q''_{fib_{n-4}}).$$

**Conjecture 19** *The size of the suffix automaton with two mismatches of the prefixes of the Fibonacci word grows according to the recursive formula*

$$q''_{fib_n} = q''_{fib_{n-1}} + 125 + 24(q''_{fib_{n-5}} - 4) + 22(q''_{fib_{n-6}}) + 11(q''_{fib_{n-5}} - 2) + 9(q''_{fib_{n-4}}).$$

Given a word $v$, Gad Landau wondered if a data structure having a size "close" to $|v|$ and that allows approximate pattern matching in time proportional to the query plus the number of occurrences exists. In the non approximate case, suffix trees and compact suffix automata do the job (cf. [12,21]). Let us see the approximate case. In [14,16,17,24,25] it is proved that for a random text $w$, the size of its compact suffix automaton with $k$ mismatches is linear times

a polylog of the size of $w$, i.e. $O(|w| \cdot \log^k |w|)$. From our experimental results it seems that the multiplicative constant hidden on the $O$ notation in this random case is very small (less than 3). By using this data structure, the time for finding the list $occ(x)$ of all occurrences of any word $x$ in the text $w$ up to $k$ mismatches is proportional to $|x| + |occ(x)|$. Therefore, for random texts the open problem of Landau has a positive answer. For prefixes of Fibonacci word our previous conjecture tells us that suffix automata do the same. In the case of words of the form $bba^n$, $a, b \in \Sigma, n \geq 1$, our experimental results have led us to the following formula describing the behaviour of the sequence of differences between two consecutive terms involving words having $n$ greater than or equal to 4: $\{a_{n+1} - a_n\} = 19 + 6 * (n - 4)$. We have experimented also on prefixes of Thue-Morse words and, even if we have not obtained a well-formed formula, we have tested that the size of the compact suffix automata with 1 mismatch obtained is less than or equal to $2 \cdot |w| \cdot \log(|w|)$. Moreover, the result is true also in the case of periodic words. We also made an exhaustive search of the worst case for binary strings of length up to 18, i.e. we looked for strings whose suffix automaton with $k$ mismatches has the greatest number of states in the case of one and two mismatches. The result was somewhat surprising. Both for one mismatch and for two mismatches one such string for any length greater than or equal to 12 in the case of one mismatch and greater than or equal to 8 in the case of two mismatches has the form $001^p01^q0$. The number $q$ seems to be always greater than $p$ and the ratio between $q$ and $p$ seems to slowly decrease and it is different in the case of one mismatch with respect to the case of two mismatches.

When we deal with non compacted suffix automaton with mismatches, the worst case within this family of strings leads to a number of states that seems to grow like $O(|w| \cdot \log^{2k}(|w|))$, where $k$ is the number of mismatches. This result is not bad at all if we consider that the language they represent has size $\Omega(|w|^{k+2})$.

If we deal with the compacted versions, results are better, but the asymptotic growth seems to be the same.

**Conjecture 20** *The compact suffix automaton with $k$ mismatches of any text $w$ has worst-case size $O(|w| \cdot \log^{2k}(|w|))$.*

It is still an open problem to find an algorithm for constructing this automaton in an efficient way.

The minimal deterministic finite automaton $S_k$ is useful for solving the problem of approximate indexing and the applications of it. Classically, an index (cf. [13]) over a fixed text $w$ is an abstract data type based on the set $Fact(w)$. Such data type is equipped with some operations that allow it to answer to the following queries. 1) Given a word $x$, say whether it belongs to $Fact(w)$

or not. If not, an index can optionally give the longest prefix of $x$ that belongs to $Fact(w)$. 2) Given $x \in Fact(w)$, find the first (respectively the last) occurrence of $x$ in $w$. 3) Given $x \in Fact(w)$, find the number of occurrences of $x$ in $w$. 4) Given $x \in Fact(w)$, find the list of all occurrences of $x$ in $w$. In the case of exact string matching, the classical data structures for indexing are suffix trees, suffix arrays, DAWGs, factor automata or their compacted versions (cf. [13]). The algorithms that use them run in a time usually independent of the size of the text or at least substantially smaller than it. The last property is required by some authors to be an essential part in the definition of an index (cf. [1]). All the operations defined for an index can easily be extended to the approximate case. But in the case of approximate string matching the problem is somehow different. We refer to [3,18,19,26] and to the references therein for a *panorama* on this subject and on approximate string matching in general. The minimal deterministic finite automaton $S_k$ introduced in this paper is useful for solving the problem of approximate indexing. More precisely, it is easy to answer queries 1) and 2), but the other questions are more complex and they can be solved by using techniques analogous to those in [17].

Moreover, if the Conjecture 20 is true and constants involved in $O$-notation are small, our data structure is useful for some classical applications of approximate indexing, as mentioned in the introduction.

Finally, we think that it is possible to connect the suffix automaton $S_k$ of the language $\mathit{Suff}(w, k, |w|)$ (without window) to the suffix automaton $S_{k,\hat{r}}$ of the language $\mathit{Suff}(w, k, \hat{r})$, with $\hat{r}$ the unique solution of the equation $r = R(w, k, r)$. More precisely, we conjecture that if $S_k$ and $S_{k,\hat{r}}$ are the suffix automata of the languages $\mathit{Suff}(w, k)$ (without window) and $\mathit{Suff}(w, k, \hat{r})$, respectively, then $|S_{k,\hat{r}}| = O(|S_k|)$. From experimental results, it seems that $S_{k,\hat{r}}$ has a greater size than $S_k$. Therefore, the algorithm described in Section 4 that accepts the language $\mathit{Suff}(w, k, \hat{r})$ seems to be more space-efficient than the automaton $S_{k,\hat{r}}$.

**Aknowledgments**

We warmly thank Gad Landau for interesting discussions on approximate text searching. We also would like to thank the anonymous referees for their suggestions leading to an improved writing of the paper.

## References


[1] A. Amir, D. Keselman, G. M. Landau, M. Lewenstein, N. Lewenstein, and M. Rodeh. Indexing and dictionary matching with one error. *LNCS*, 1663:181–190, 1999.

[2] A. Amir, D. Keselman, G. M. Landau, M. Lewenstein, N. Lewenstein, and M. Rodeh. Indexing and dictionary matching with one error. *Journal of Algorithms*, 37:309–325, 2000.

[3] R. Baeza-Yates, G. Navarro, E. Sutinen, and J. Tarhio. Indexing methods for approximate string matching. *IEEE Data Engineering Bulletin*, 24(4):19–27, 2001. Special issue on Managing Text Natively and in DBMSs. Invited paper.

[4] A. Blumer, J. Blumer, A. Ehrenfeucht, D. Haussler, and R. M. McConnell. Linear size finite automata for the set of all subwords of a word - an outline of results. *Bulletin of the EATCS*, 21:12–20, 1983.

[5] A. Blumer, J. Blumer, D. Haussler, A. Ehrenfeucht, M. Chen, and J. Seiferas. The smallest automaton recognizing the subwords of a text. *Theoretical Computer Science*, 40:31–55, 1985.

[6] A. L. Buchsbaum, M. T. Goodrich, and J. Westbrook. Range searching over tree cross products. In *ESA 2000*, volume 1879, pages 120–131, 2000.

[7] A. Carpi and A. de Luca. Words and special factors. *Theoretical Computer Science*, 259:145–182, 2001.

[8] E. Chávez and G. Navarro. A metric index for approximate string matching. In *LATIN 2002: Theoretical Informatics: 5th Latin American Symposium, Cancun, Mexico, April 3-6, 2002. Proceedings*, volume 2286 of *LNCS*, pages 181–195, 2002.

[9] R. Cole, L. Gottlieb, and M. Lewenstein. Dictionary matching and indexing with errors and don't cares. In *Proceedings of Annual ACM Symposium on Theory of Computing (STOC 2004)*, 2004.

[10] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms and Java CD-ROM.* McGraw-Hill, 2003.

[11] M. Crochemore. Transducers and repetitions. *Theoretical Computer Science*, 45(1):63–86, 1986.

[12] M. Crochemore. Reducing space for index implementation. *Theoretical Computer Science*, 292(1):185–197, 2003.

[13] M. Crochemore, C. Hancart, and T. Lecroq. *Algorithms on strings.* Cambridge University Press, 2007.

[14] C. Epifanio, A. Gabriele, and F. Mignosi. Languages with mismatches and an application to approximate indexing. In *Proceedings of DLT05*, volume 3572 of *LNCS*, pages 224–235, 2005.

[15] C. Epifanio, A. Gabriele, F. Mignosi, A. Restivo, and M. Sciortino. Languages with mismatches. *Theoretical Computer Science*, 385:152–166, 2007.

[16] A. Gabriele. *Combinatorics on words with mismatches, algorithms and data structures for approximate indexing with applications*. PhD thesis, University of Palermo, 2004.

[17] A. Gabriele, F. Mignosi, A. Restivo, and M. Sciortino. Indexing structure for approximate string matching. In *Proceedings of CIAC'03*, volume 2653 of *LNCS*, pages 140–151, 2003.

[18] Z. Galil and R. Giancarlo. Data structures and algorithms for approximate string matching. *Journal of Complexity*, 24:33–72, 1988.

[19] D. Gusfield. *Algorithms on Strings, Trees, and Sequences*. Cambridge University Press, 1997.

[20] T. N. D. Huynh, W. K. Hon, T. W. Lam, and W. K. Sung. Approximate string matching using compressed suffix arrays. In *Proceedings of CPM 2004*, volume 3109 of *LNCS*, pages 434–444.

[21] S. Inenaga, H. Hoshino, A. Shinohara, M. Takeda, S. Arikawa, G. Mauri, and G. Pavesi. On-line construction of compact directed acyclic word graphs. *Discrete Applied Mathematics*, 146(2):156–179, 2005.

[22] G. M. Landau and U. Vishkin. Efficient string matching with $k$ mismatches. *Theoretical Computer Science*, 43 (2–3):239–249, 1986.

[23] M. Lothaire. *Combinatorics on Words*, volume 17 of *Encyclopedia of Mathematics*. Cambridge University Press, 1983.

[24] M. G. Maass and J. Nowak. A new method for approximate indexing and dictionary lookup with one error. *Information Processing Letters*, 96(5):185–191, 2005.

[25] M. G. Maass and J. Nowak. Text indexing with errors. In *Proceedings of the 16th Annual Symposium on Combinatorial Pattern Matching (CPM 2005)*, LNCS 3537, pages 21–32, 2005.

[26] G. Navarro. A guided tour to approximate string matching. *ACM Computing Surveys*, 33(1):31–88, 2001.

[27] D. Sankoff and J. B. Kruskal. *Time Warps, String Edits, and Macromolecules: The Theory and Practice of Sequence Comparison*. Addison-Wesley Publishing Co., Reading, Massachusetts, 1983.

[28] W. Szpankowski. *Average Case Analysis of Algorithms on Sequences*. John Wiley & Sons, 2001.