Text Compression Using Antidictionaries*

M. Crochemore[†], F. Mignosi[‡], A. Restivo[‡], S. Salemi[‡]

June 10, 1998

Abstract

We give a new text compression scheme based on Forbidden Words ("antidictionary"). We prove that our algorithms attain the entropy for equilibrated binary sources. One of the main advantage of this approach is that it produces very fast decompressors. A second advantage is a synchronization property that is helpful to search compressed data and to parallelize the compressor. Our algorithms can also be presented as "compilers" that create compressors dedicated to any previously fixed source. The techniques used in this paper are from Information Theory and Finite Automata; as a consequence, this paper shows that Formal Language Theory (in particular Finite Automata Theory) can be useful in Data Compression.

Keywords: data compression, information theory, finite automaton, forbidden word, pattern matching.

1 Introduction

We present a simple text compression method called DCA (Data Compression with Antidictionaries) that uses some "negative" information about the text, which is described in terms of antidictionaries. Contrary to other methods that make use, as a main tool, of dictionaries, *i.e.*, particular sets of words occurring as factors in the text (cf. [6], [13], [17], [19] and [20]), our method takes advantage from words that do not occur as factor in the text, *i.e.*, that are forbidden. Such sets of words are called here antidictionaries.

Let w be a text on the binary alphabet $\{0, 1\}$ and let AD be an antidictionary for w. By reading the text w from left to right, if at a certain moment the current prefix v of the text admits as suffix a word u' such that $u = u'x \in AD$ with $x \in \{0, 1\}$, *i.e.*, u is forbidden, then surely the letter following v in the text cannot be x and, since the alphabet is binary, it is the letter $y \neq x$. In other

*Web page at URL `http://www-igm.univ-mlv.fr/~mac/DCA.html`

[†]Institut Gaspard-Monge

[‡]Università di Palermo

terms, we know in advance the next letter y, that turns out to be redundant or predictable. The main idea of our method is to eliminate redundant letters in order to achieve compression. The decoding algorithm recovers the text w by predicting the letter following the current prefix v of w already decompressed.

The method here proposed presents some analogies with ideas discussed by C. Shannon at the very beginning of Information Theory. In [18] Shannon designed psychological experiments in order to evaluate the entropy of English. One of such experiments was about the human ability to reconstruct an english text where some characters were erased. Actually our compression methods erases some characters and the decompression reconstruct them.

We prove that the compression rate of our compressor reaches the entropy almost surely, provided that the source is equilibrated and produced from a finite antidictionary. This type of source approximates a large class of sources, and consequently, a variant of the basic scheme gives an optimal compression for them. The idea of using antidictionaries is founded on the fact that there exists a topological invariant for Dynamical Systems based on forbidden words and independent of the entropy [4].

The use of the antidictionary AD in coding and decoding algorithms requires that AD must be structured in order to answer to the following query on a word v: does there exists a word $u = u'x$, $x \in \{\mathtt{0}, \mathtt{1}\}$, in AD such that u' is a suffix of v? In the case of positive answer the output should also include the letter y defined by $y \neq x$. One of the main features of our method is that we are able to implement efficiently finite antidictionaries in terms of finite automata. This leads to efficient and fast compression and decompression algorithms, which can be realized by sequential transducers (generalized sequential machines). This is especially relevant for fixed sources. It is then comparable to the fastest compression methods because the basic operation at compression and decompression time is just table lookup.

A central notion of the present method is that of minimal forbidden words, which allows to reduce the size of antidictionaries. This notion has also some interesting combinatorial properties. Our compression method includes algorithms to compute antidictionaries, algorithms that are based on the above combinatorial properties and that are described in details in [8] and [9].

The compression method shares also an interesting synchronization property, in the case of finite antidictionaries. It states that the encoding of a block of data does not depend on the left and right contexts except for a limited-size prefix of the encoded block. This is helpful to search compressed data, which is not a common feature of other compression methods. The same property allows to design efficient parallel compression algorithms.

The paper is organized as follows.

In Section 2 we give the definition of Forbidden Words and of antidictionaries. We describe DCA, our text compression and decompression algorithms (binary oriented) assuming that the antidictionary is given and that we can perform special kinds of queries on this antidictionary.

In Section 3 we describe a data structure for finite antidictionaries that allows to answer in efficient way to the queries needed by our compression and decompression algorithms; we show how to implement it given a finite antidictionary. The compression is also described in terms of transducers, which is valid only in the case of rational antidictionaries. We end the section by proving the synchronizion property.

In Section 4 we evaluate the compression rate of our compression algorithm relative to a given antidictionary.

In Section 5 we show how to construct antidictionaries for single words and sources. As a consequence we obtain a family of linear time optimal algorithms for text compression that are universal for equilibrated sources generated from antidictionaries.

We report in Section 6 experimental results made with a dynamic prototype of DCA. It shows that the compression rate is similar to those of most common compressors.

We consider possible generalizations in the conclusion (Section 7).

2 Forbidden Words and the Basic Algorithms

Let us first introduce the main ideas of our algorithm. Let w be a finite binary word and let $F(w)$ be the set of factors of w.

For instance, if $w = \mathtt{01001010010}$

$$F(w) = \{\varepsilon, \mathtt{0}, \mathtt{1}, \mathtt{00}, \mathtt{01}, \mathtt{10}, \mathtt{001}, \mathtt{010}, \mathtt{100}, \mathtt{101}, \ldots\}$$

where ε denotes the empty word.

Let us take some words in the complement of $F(w)$, *i.e.*, let us take some words that are not factors of w and that we call *forbidden*. The set of such words AD is called an *antidictionary* of the language $F(w)$. Antidictionaries can be finite as well infinite.

For instance, if $w = \mathtt{01001010010}$ the words $\mathtt{11}$, $\mathtt{000}$, and $\mathtt{10101}$ are forbidden and the set $AD = \{\mathtt{000}, \mathtt{10101}, \mathtt{11}\}$ is an antidictionary of $F(w)$.

The compression algorithm treats the input word in an online manner. At a certain moment in this process we have read the word v prefix of w. If there exists a word $u = u'x$, $x \in \{\mathtt{0}, \mathtt{1}\}$, in the antidictionary AD such that u' is a suffix of v, then surely the letter following v cannot be x, *i.e.*, the next letter is y, $y \neq x$. In other words, we know in advance the next letter y that turns out to be "redundant" or predictable. Remark that this argument works only in the case of binary alphabets.

The main idea in the algorithm we describe is to eliminate redundant letters in order to achieve compression.

In what follows we first describe the compression algorithm, ENCODER and then the decompression algorithm, DECODER. The word to be compressed is noted $w = a_1 \cdots a_n$ and its compressed version is denoted by $\gamma(w)$.

ENCODER (anti-dictionary AD, word $w \in \{\mathtt{0}, \mathtt{1}\}^*$)
$v \leftarrow \varepsilon$; $\gamma \leftarrow \varepsilon$;
for $a \leftarrow$ first to last letter of w
  $v \leftarrow v.a$;
  if for any suffix u' of v, $u'\mathtt{0}, u'\mathtt{1} \notin AD$
    $\gamma \leftarrow \gamma.a$;
return $(|v|, \gamma)$;

As an example, let us run the algorithm ENCODER on the string $w =$ $\mathtt{01001010010}$ with the antidictionary $AD = \{\mathtt{000}, \mathtt{10101}, \mathtt{11}\}$. The steps of the treatment are described in the next array by the current values of the prefix $v_i = a_1 \cdots a_i$ of w that has been just considered and of the output $\gamma(w)$. In the case of positive answer to the query to the antidictionary AD, the array also indicates the value of the corresponding forbidden word u. The number of times the answer is positive in a run corresponds to the number of bits erased.

ε	$\gamma(w) = \varepsilon$	
$v_1 = \mathtt{0}$	$\gamma(w) = \mathtt{0}$	
$v_2 = \mathtt{01}$	$\gamma(w) = \mathtt{01}$	$u = \mathtt{11} \in AD$
$v_3 = \mathtt{010}$	$\gamma(w) = \mathtt{01}$	
$v_4 = \mathtt{0100}$	$\gamma(w) = \mathtt{010}$	$u = \mathtt{000} \in AD$
$v_5 = \mathtt{01001}$	$\gamma(w) = \mathtt{010}$	$u = \mathtt{11} \in AD$
$v_6 = \mathtt{010010}$	$\gamma(w) = \mathtt{010}$	
$v_7 = \mathtt{0100101}$	$\gamma(w) = \mathtt{0101}$	$u = \mathtt{11} \in AD$
$v_8 = \mathtt{01001010}$	$\gamma(w) = \mathtt{0101}$	$u = \mathtt{10101} \in AD$
$v_9 = \mathtt{010010100}$	$\gamma(w) = \mathtt{0101}$	$u = \mathtt{000} \in AD$
$v_{10} = \mathtt{0100101001}$	$\gamma(w) = \mathtt{0101}$	$u = \mathtt{11} \in AD$
$v_{11} = \mathtt{01001010010}$	$\gamma(w) = \mathtt{0101}$	

Remark that the function γ is not injective. For instance $\gamma(\mathtt{010010100}) = \gamma(\mathtt{0100101001}) = \mathtt{0101}$. In order to have an injective mapping we can consider the function $\gamma'(w) = (|w|, \gamma(w))$. In this case we can reconstruct the original word w from both $\gamma'(w)$ and the antidictionary.

The decoding algorithm works as follow. The compressed word is $\gamma(w) = b_1 \cdots b_h$ and the length of w is n. The algorithm recovers the word w by predicting the letter following the current prefix v of w already decompressed. If there exists a word $u = u'x$, $x \in \{\mathtt{0}, \mathtt{1}\}$, in the antidictionary AD such that u' is a suffix of v, then, the output letter is y, $y \neq x$. Otherwise, the next letter is read from the input γ.

Decoder (anti-dictionary AD, integer n, word $\gamma \in \{\mathtt{0},\mathtt{1}\}^*$)

```
v ← ε;
while |v| < n
    if for some u' suffix of v and x ∈ {0,1}, u'x ∈ AD
        v ← v · ¬x;
    else
        b ← next letter of γ;
        v ← v · b;
return (v);
```

The antidictionary AD must be structured in order to answer to the following query on a word v: does there exist a word $u = u'x$, $x \in \{\mathtt{0},\mathtt{1}\}$, in AD such that u' is a suffix of v? In case of a positive answer the output should also include the letter y defined by $y \neq x$.

The method presented here brings to mind some ideas proposed by C. Shannon at the very beginning of Information Theory. In [18] Shannon designed psychological experiments in order to evaluate the entropy of English. One of such experiments was about the human ability to reconstruct an english text where some characters were erased. Actually our compression methods erases some characters and the decompression reconstruct them. For instance in previous example the input string is

$$\mathtt{0}\mathtt{1}\bar{\mathtt{0}}\mathtt{0}\bar{\mathtt{1}}\bar{\mathtt{0}}\mathtt{1}\bar{\mathtt{0}}\bar{\mathtt{0}}\bar{\mathtt{1}}\bar{\mathtt{0}},$$

where bars indicate the letters erased in the compression algorithm.

In this approach the encoder must send to the decoder the length of the word $|w|$, the compressed word $\gamma(w)$ as well the antidictionary in the case the decoder has not yet a copy of the antidictionary.

In order to have a good compression rate we need to minimize in particular the size of the antidictionary. Remark that if there exists a forbidden word $u = u'x$, $x \in \{\mathtt{0},\mathtt{1}\}$ in the antidictionary such that u' is also forbidden then our algorithm will never use this word u in the algorithms. So that we can erase this word from the antidictionary without any loss for the compression of w.

This argument leads to introduce the notion of *minimal forbidden word* with respect to a factorial language L, notion that is discussed in the next section.

3 Implementation of Finite Antidictionaries

The queries on the antidictionary required by the algorithm of the previous section are realized as follows. We build the deterministic automaton accepting the words having no factor in the antidictionary. Then, while reading the text to encode, if a transition leads to a sink state, the output is the other letter. What remains to explain is how the automaton is built from the antidictionary.

The wanted automaton accepts a factorial language L. Recall that a language L is factorial if L satisfies the following property: for any words, u, v, $uv \in L \Rightarrow u \in L$ and $v \in L$.

The complement language $L^c = A^* \setminus L$ is a (two-sided) ideal of A^*. Denoting by $MF(L)$ the base of this ideal, we have $L^c = A^* MF(L) A^*$. The set $MF(L)$ is called the set of *minimal forbidden words* for L. A word $v \in A^*$ is forbidden for the factorial language L if $v \notin L$, which is equivalent to say that v occurs in no word of L. In addition, v is minimal if it has no proper factor that is forbidden.

One can note that the set $MF(L)$ uniquely characterizes L, just because

$$L = A^* \setminus A^* MF(L) A^*. \tag{1}$$

Indeed, there is a duality between factorial and anti-factorial languages, because we also have the equality:

$$MF(L) = AL \cap LA \cap (A^* \setminus L). \tag{2}$$

As a consequence of both equalities (1) and (2) we get the following proposition.

Proposition 1 *For a factorial language L, languages L and $MF(L)$ are simultaneously rational, that is, $L \in Rat(A^*)$ iff $MF(L) \in Rat(A^*)$.*

The set $MF(L)$ is an *anti-factorial language* or a *factor code*, which means that it satisfies: $\forall u, v \in MF(L)\ u \neq v \Longrightarrow u$ is not a factor of v, property that comes from the minimality of words of $MF(L)$.

We introduce a few more definitions.

Definition 1 *A word $v \in A^*$* avoids the set *M, $M \subseteq A^*$, if no word of M is a factor of v, (i.e., if $v \notin A^* M A^*$). A language L* avoids *M if every word of L avoid M.*

From the definition of $MF(L)$, it readily comes that L is the largest (according to the subset relation) factorial language that avoids $MF(L)$. This shows that for any anti-factorial language M there exists a unique factorial language $L(M)$ for which $M = MF(L)$. The next remark summarizes the relation between factorial and anti-factorial languages.

Remark 1 *There is a one-to-one correspondence between factorial and anti-factorial languages. If L and M are factorial and anti-factorial languages respectively, both equalities hold: $MF(L(M)) = M$ and $L(MF(L)) = L$.*

Finally, with a finite anti-factorial language M we associate the finite automaton $\mathcal{A}(M)$ as described below. The automaton is deterministic and complete, and, as shown at the end of the section by Theorem 1, it accepts the language $L(M)$.

The automaton $\mathcal{A}(M)$ is the tuple (Q, A, i, T, F) where

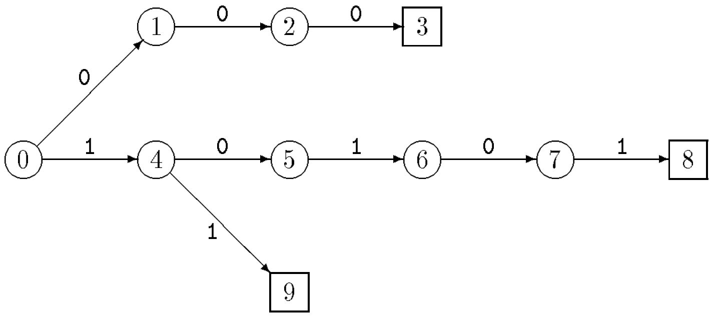

Figure 1: Trie of the factor code $\{\texttt{000}, \texttt{10101}, \texttt{11}\}$. Squares represent terminal states.

- the set Q of states is $\{w \mid w \text{ is a prefix of a word in } M\}$,
- A is the current alphabet,
- the initial state i is the empty word ε,
- the set T of terminal states is $Q \setminus M$.

States of $\mathcal{A}(M)$ that are words of M are sink states. The set F of transitions is partitioned into the three (pairwise disjoint) sets F_1, F_2, and F_3 defined by:

- $F_1 = \{(u, a, ua) \mid ua \in Q, a \in A\}$ (forward edges or tree edges),
- $F_2 = \{(u, a, v) \mid u \in Q \setminus M, a \in A, ua \notin Q, v \text{ longest suffix of } ua \text{ in } Q\}$ (backward edges),
- $F_3 = \{(u, a, u) \mid u \in M, a \in A\}$ (loops on sink states).

The transition function defined by the set F of arcs of $\mathcal{A}(M)$ is noted δ.

The next result is proved in [8].

Theorem 1 *For any anti-factorial language M, $\mathcal{A}(M)$ accepts the language $L(M)$.*

The above definition of $\mathcal{A}(M)$ turns into the algorithm below, called L-AUTOMATON, that builds the automaton from a finite anti-factorial set of words. The input is the trie $\mathcal{T}$ that represents M. It is a tree-like automaton accepting the set M and, as such, it is noted (Q, A, i, T, δ').

In view of Equality 1, the design of the algorithm remains to adapt the construction of a pattern matching machine (see [1] or [7]). The algorithm uses a function f called a *failure function* and defined on states of $\mathcal{T}$ as follows. States of the trie $\mathcal{T}$ are identified with the prefixes of words in M. For a state au ($a \in A$, $u \in A^*$), $f(au)$ is $\delta'(i, u)$, quantity that may happen to be u itself. Note that $f(i)$ is undefined, which justifies a specific treatment of the initial state in the algorithm.

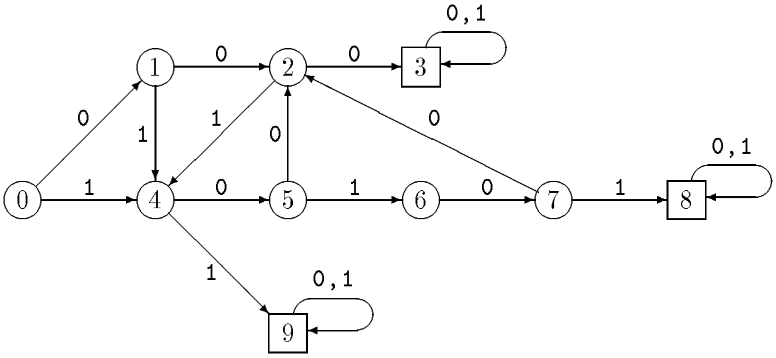

Figure 2: Automaton accepting the words that avoid the set $\{\texttt{000}, \texttt{10101}, \texttt{11}\}$. Squares represent non-terminal states (sink states).

```
L-AUTOMATON (trie T = (Q, A, i, T, δ'))
for each a ∈ A
    if δ'(i, a) defined
        set δ(i, a) = δ'(i, a);
        set f(δ(i, a)) = i;
    else
        set δ(i, a) = i;
for each state p ∈ Q \ {i} in width-first search and each a ∈ A
    if δ'(p, a) defined
        set δ(p, a) = δ'(p, a);
        set f(δ(p, a)) = δ(f(p), a);
    else if p ∉ T
        set δ(p, a) = δ(f(p), a);
    else
        set δ(p, a) = p;
return (Q, A, i, Q \ T, δ);
```

Example. Figure 1 displays the trie that accepts $M = \{\texttt{000}, \texttt{10101}, \texttt{11}\}$. It is an anti-factorial language. The automaton produced from the trie by algorithm L-AUTOMATON is shown in Figure 2. It accepts all the words avoiding M.

Theorem 2 *Let $\mathcal{T}$ be the trie of an anti-factorial language M. Algorithm* L-AUTOMATON *builds a complete deterministic automaton accepting $L(M)$.*

Proof. The automaton produced by the algorithm has the same set of states as the input trie. It is clear that the automaton is deterministic and complete.

Let $u \in A^+$ and $p = \delta(i, u)$. A simple induction on $|u|$ shows that the word corresponding to $f(p)$ is the longest proper suffix of u that is a prefix of some word in M. This notion comes up in the definition of the set of transitions F_2

in the automaton $\mathcal{A}(M)$. Therefore, the rest of the proof just remains to check that instructions implement the definition of $\mathcal{A}(M)$. $\bowtie$

Theorem 3 *Algorithm* L-AUTOMATON *runs in time* $O(|Q| \times |A|)$ *on input* $\mathcal{T} = (Q, A, i, T, \delta')$ *if transition functions are implemented by transition matrices.*

Proof. If transition functions δ and δ' are implemented by transition matrices, access to or definition of $\delta(p, a)$ or $\delta'(p, a)$ (p state, $a \in A$) are realized in constant amount of time. The result follows immediately. $\bowtie$

The algorithm L-AUTOMATON can be adapted to test whether $\mathcal{T}$ represents an anti-factorial set, to generate the trie of the anti-factorial language associated with a set of words, or even to built the automaton associated with the anti-factorial language corresponding to any set of words.

Transducers

From the automaton $\mathcal{A}(M)$ we can easily construct a (finite-state) transducer $\mathcal{B}(M)$ that realizes the compression algorithm ENCODER, *i.e.*, that computes the function γ.

The input part of $\mathcal{B}(M)$ coincides with $\mathcal{A}(M)$ and the output is given as follows: if a state of $\mathcal{A}(M)$ has two outgoing edges, then the output labels of these edges coincide with their input label; if a state of $\mathcal{A}(M)$ has only one outgoing edge, then the output label of this edge is the empty word.

The transducer $\mathcal{B}(M)$ works as follows on an input string w. Consider the (unique) path in $\mathcal{B}(M)$ corresponding to w. The letters of w that correspond to an edge that is the unique outgoing edge of a given state are erased; other letters are unchanged.

We can then state the following theorem.

Theorem 4 *Algorithm* ENCODER *is realized by a sequential transducer (generalized sequential machine).*

As to concern the algorithm DECODER, remark (see Section 2) that the function γ is not injective and that we need some additional information, for instance the length of the original uncompressed word, in order to reconstruct it without ambiguity. We show that DECODER can be realized by the same transducer as above, by interchanging input and output labels (denote it by $B'(M)$), with a supplementary instruction to stop the decoding.

Let $Q = Q_1 \cup Q_2$ be a partition of the set of states Q, where Q_i is the set of states having i outpoing edges ($i = 1, 2$). For any $q \in Q_1$, define $p(q) = (q, q_1, \ldots, q_r)$ as the unique path in the tranducer for which $q_j \in Q_1$ for $j < r$ and $q_r \in Q_2$.

Given an input word $v = b_1 b_2 \dots b_m$, there exists in $B'(M)$ a unique path $i, q_1, \dots, q_{m'}$ such that $q_{m'-1} \in Q_2$ and the transition from $q_{m'-1}$ to q'_m correspond to the input letter b_m

If $q_{m'} \in Q_2$, then the output word corresponding to this path in $B'(M)$ is the unique word w such that $\gamma(w) = v$.

If $q_{m'} \in Q_1$, then we can stop the run of the decoding algorithm realized by $B'(M)$ in any state $q \in p(q_{m'})$, and, for different states, we obtain different decoding. So, we need a supplementary information (for instance the length of the original uncompressed word) to perform the decoding. In this sense we can say that $B'(M)$ realizes sequentially the algorithm DECODER.

The constructions and the results given above can be generalized also to the case of rational antidictionaries, or, equivalently, when the set of words "produced by the source" is a rational language. In these cases it is not, in a strict sense, necessary to introduce explicitely antidictionaries and all the methods can be presented in terms of automata and tranducers, as above. Remark however that the presentation given in Section 2 in terms of antidictionaries is more general, since it includes the non rational case. Moreover, even in the finite case, the construction of automata and transducers from a fixed text, given in the next section, makes an explicit use of the notion of minimal forbidden words and of antidictionaries.

A Synchronization Property

In the sequel we prove a synchronization property of automata built from finite antidictionaries, as described above. This property also "characterizes" in some sense finite antidictionaries. This property is a classical one and it is of fundamental importance in practical applications.

We start with a definition.

Definition 2 *Given a deterministic finite automaton $\mathcal{A}$, we say that a word $w = a_1 \cdots a_n$ is synchronizing for $\mathcal{A}$ if, whenever w represents the label of two paths $(q_1, a_1, q_2) \cdots (q_n, a_n, q_{n+1})$ and $(q'_1, a_1, q'_2) \cdots (q'_n, a_n, q'_{n+1})$ of length n, then the two ending states q_{n+1} and q'_{n+1} are equal.*

If $L(\mathcal{A})$ is factorial, any word that does not belong to $L(\mathcal{A})$ is synchronizing. Clearly in this case synchronizing words in $L(\mathcal{A})$ are much more interesting.

Remark also that, since $\mathcal{A}$ is deterministic, if w is synchronizing for $\mathcal{A}$, then any word $w' = wv$ that has w as prefix is also synchronizing for $\mathcal{A}$.

Definition 3 *A deterministic finite automaton $\mathcal{A}$ is local if there exists an integer n such that any words of length n is synchronizing. Automaton $\mathcal{A}$ is also called n-local.*

Remark that if $\mathcal{A}$ is n-local then it is m-local for any $m \geq n$.

Given a finite antifactorial language M, let $\mathcal{A}(M)$ be the automaton associated with M as described in Section 4. Recall that it has no sink state, that all states are terminal, and that $L(\mathcal{A}(M))$ is factorial.

Theorem 5 *Let M be a finite antifactorial antidictionary and let n be the length of the longest word in M. Then automaton $\mathcal{A}(M)$ associated to M is $(n-1)$-local.*

Proof. Let $u = a_1 \cdots a_{n-1}$ be a word of length $n-1$. We have to prove that u is synchronizing. Suppose that there exist two paths $(q_1, a_1, q_2) \cdots (q_{n-1}, a_{n-1}, q_n)$ and $(q'_1, a_1, q'_2) \cdots (q'_{n-1}, a_{n-1}, q'_n)$ of length $n-1$ labeled by u. We have to prove that the two ending states q_n and q'_n are equal. Recall that states of $\mathcal{A}$ are words, and, more precisely they are the proper prefixes of words in M. A simple induction on i, $1 \leq i \leq n$ shows that q_i (respectively q'_i) "is" the longest suffix of the word $q_1 a_1 \cdots a_i$ (respectively $q'_1 a_1 \cdots a_i$) that is also a "state", *i.e.*, a proper prefix of a word in M. Hence q_n (respectively q'_n) is the longest suffix of the word $q_1 u$ (respectively $q'_1 u$) that is also a proper prefix of a word in M. Since all proper prefixes of words in M have length at most $n-1$, both q_n and q'_n have length at most $n-1$. Since u has length $n-1$, both they are the longest suffix of u that is also a proper prefix of a word in M, *i.e.*, they are equal. $\bowtie$

The previous theorem admits a "converse" that we state without proof and that shows that locality characterizes in some sense finite antidictionaries (cf. Propositions 2.8 and 2.14 of [3]).

Theorem 6 *If automaton $\mathcal{A}$ is local and $L(\mathcal{A})$ is a factorial language then there exists a finite antifactorial language M such that $L(\mathcal{A}) = L(M)$.*

Let M be an antifactorial antidictionary and let n be the length of the longest word in M. Let also $w = w_1 u v w_2 \in L(M)$ with $|u| = n-1$ and let $\gamma(w) = y_1 y_2 y_3$ be the word produced by our encoder of Section 2 with input M and w. The word y_1 is the word produced by our encoder after processing $w_1 u$, the word y_2 is the word produced by our encoder after processing v and the word y_3 is the word produced by our encoder after processing w_2.

The proof of next theorem is an an easy consequence of previous definitions and of the statement of Theorem 5.

Theorem 7 *The word y_2 depends only on the word uv and it does not depend on the contexts of it, w_1 and w_2.*

The property stated in the theorem has an interesting consequence for the design of pattern matching algorithms on words compressed by the algorithm Encoder. It implies that, to search the compressed word for a pattern, it is not necessary to decode the whole word. Just a limited left context of an occurrence of the pattern needs to be processed (cf. [10]). This is not a common feature of other compression methods. They have to split the input to get the same advantage, but this may weaken the efficiency of the final compression algorithm.

The same property allows the design of highly parallizable compression algorithms. The idea is that the compression can be performed independently and in parallel on any block of data. If the text to be compressed is parsed into blocks of data in such a way that each block overlaps the next block by a length not smaller than the length of the longest word in the antidictionary, then it is possible to run the whole compression process.

4 Efficiency

In this section we evaluate the efficiency of our compression algorithm relatively to a source corresponding to the finite antidictionary M.

Indeed, the antidictionary M defines naturally a source $S(M)$ in the following way. Let $\mathcal{A}(M)$ be the automaton constructed in the previous section and that recognizes the language $L(M)$, and let us eliminate the sink states and edges going to them. Since there is no possibility of misunderstanding, we denote the resulting automaton by $\mathcal{A}(M)$ again. To avoid trivial cases we suppose that in this automaton all the states have at least one outgoing edge. Recall that, since our algorithms work on binary alphabets, all the states have at most two outgoing edges.

For any state of $\mathcal{A}(M)$ with only one outgoing edge we give to this edge probability 1. For any state of $\mathcal{A}(M)$ with two outgoing edge we give to these edges probability $1/2$. This defines a deterministic (or unifilar, cf. [2]) Markov source, denoted $S(M)$. A binary Markov source with this probability distribution is called an *equilibrated source.*

Remark that our compression algorithm is defined exactly for all the words "emitted" by $S(M)$.

In what follows we suppose that the graph of the source S, *i.e.*, the graph of automaton $\mathcal{A}(M)$ is strongly connected. The results that we prove can be extended to the general case by using standard techniques of Markov Chains (cf. [2] and [14]).

Recall (cf. Theorem 6.4.2 of [2]) that the entropy $H(S)$ of a deterministic markov source S is

$$H(S) = -\Sigma_{i,j=1}^{n} \mu_i \gamma_{i,j} \ log_2(\gamma_{i,j}),$$

where $(\gamma_{i,j})$ is the stochastic matrix of S and $(\mu_1, \cdots, \mu_n)$ is the stationary distribution of S.

We first start with two preliminary lemmas.

Lemma 1 *The entropy of an equilibrated source S is given by $H(S) = \Sigma_{i \in D} \mu_i$ where D is the set of all states that have two outgoing edges.*

Proof. By definition

$$H(S) = -\Sigma_{i,j=1}^{n} \mu_i \gamma_{i,j} \ log_2(\gamma_{i,j}).$$

If i is a state with only one outgoing edge, by definition this edge must have probability 1. Then $\Sigma_j \mu_i \gamma_{i,j} \ log_2(\gamma_{i,j})$ reduces to $\mu_i \ log_2(1)$, that is equal to 0. Hence

$$H(S) = -\Sigma_{i \in D} \Sigma_{j=1}^{n} \mu_i \gamma_{i,j} \ log_2(\gamma_{i,j}).$$

Since from each $i \in D$ there are exactly two outgoing edges having each probability $1/2$, one has

$$H(S) = -\Sigma_{i \in D} 2\mu_i (1/2) \ log_2(1/2) = \Sigma_{i \in D} \mu_i$$

as stated. ⋈

Lemma 2 *Let $w = a_1 \cdots a_m$ be a word in $L(M)$ and let $q_1 \cdots q_{m+1}$ be the sequence of states in the path determined by w in $\mathcal{A}(M)$ starting from the initial state. The length of $\gamma(w)$ is equal to the number of states q_i, $i = 1, \ldots, m$, that belong to D, where D is the set of all states that have two outgoing edges.*

Proof. The statement is straightforward from the description of the compression algorithm and the implementation of the antidictionary with automaton $\mathcal{A}(M)$. ⋈

Next lemma reports a well known "large deviation" result (cf. Theorem 1.4.3 of [12]).

Let $\mathbf{q}= q_1, \cdots q_m$ be the sequence of m states of a path of $\mathcal{A}(M)$ and let $L_{m,i}(\mathbf{q})$ be the frequency of state q_i in this sequence, *i.e.*, $L_{m,i}(\mathbf{q}) = m_i/m$, where m_i is the number of occurrences of q_i in the sequences $\mathbf{q}$. Let also

$$X_m(\epsilon) = \{ \ \mathbf{q} \ \mid \mathbf{q} \text{ has } m \text{ states and } max_i \ |L_{m,i}(\mathbf{q}) - \mu_i| \geq \epsilon \},$$

where $\mathbf{q}$ represents a sequence of m states of a path in $\mathcal{A}(M)$.

In other words, $X_m(\epsilon)$ is the set of all sequences of states representing path in $\mathcal{A}(M)$ that "deviate" at least of ϵ in at least one state q_i from the theoretical frequency μ_i.

Lemma 3 *For any $\epsilon > 0$, the set $X_m(\epsilon)$ satisfies the equality*

$$\lim \frac{1}{m} \log_2 \Pr(X_m(\epsilon)) = -c(\epsilon),$$

where $c(\epsilon)$ is a positive constant depending on ϵ.

We now state and prove the main theorem of this section. We prove that for any ϵ the probability that the compression rate $\tau(v) = |\gamma(v)|/|v|$ of a string of length n is greater than $H(S(M)) + \epsilon$, goes exponentially to zero. Hence, as a corollary, almost surely the compression rate of an infinite sequence emitted by $S(M)$ reaches the entropy $H(S(M))$, that is the best possible result.

Let $K_m(\epsilon)$ be the set of words w of length m such that the compression rate $\tau(v) = |\gamma(v)|/|v|$ is greater than $H(S(M)) + \epsilon$.

Theorem 8 *For any $\epsilon > 0$ there exist a real number $r(\epsilon)$, $0 < r(\epsilon) < 1$, and an integer $\overline{m}(\epsilon)$ such that for any $m > \overline{m}(\epsilon)$, $\Pr(K_m(\epsilon)) \leq r(\epsilon)^m$.*

Proof. Let w be a word of length m in the language $L(M)$ and let $q_1, \cdots, q_{m+1}$ be the sequence of states in the path determined by w in $\mathcal{A}(M)$ starting from the initial state. Let $\mathbf{q}= (q_1, \cdots, q_m)$ be the sequence of the first m states. We know, by Lemma 2, that the length of $\gamma(w)$ is equal to the number of states q_i, $i = 1 \cdots m$, in $\mathbf{q}$ that belong to D, where D is the set of all states having two outgoing edges.

If w belog to $K_m(\epsilon)$, *i.e.*, if the compression rate $\tau(v) = |\gamma(v)|/|v|$ is greater than $H(S(M)) + \epsilon$, then there must exists an index j such that $L_{m,j}(\mathbf{q}) > \mu_j + \epsilon/d$, where d is the cardinality of the set D. In fact, if for all j, $L_{m,j}(\mathbf{q}) \leq \mu_j + \epsilon/|D|$ then, by definitions and by Lemma 1,

$$\tau(v) = \Sigma_{j \in D} L_{m,j}(\mathbf{q}) \leq \Sigma_{j \in D} \mu_j + \epsilon = H(S(M)) + \epsilon,$$

a contradiction. Therefore the sequence of states $\mathbf{q}$ belongs to $X_m(\epsilon/d)$.

Hence $Pr(K_m(\epsilon)) \leq Pr(X_m(\epsilon/d))$.

By Lemma 3, there exists an integer $\overline{m}(\epsilon)$ such that for any $m > \overline{m}(\epsilon)$ one has

$$\frac{1}{m} log_2 Pr(X_m(\frac{\epsilon}{d})) \leq -\frac{1}{2} c(\frac{\epsilon}{d}).$$

Then $Pr(K_m(\epsilon)) \leq 2^{-(1/2)c(\epsilon/d)m}$. If we set $r(\epsilon) = 2^{-(1/2)c(\epsilon/d)}$, the statement of the theorem follows. $\bowtie$

Corollary 1 *The compression rate $\tau(\mathbf{x})$ of an infinite sequence $\mathbf{x}$ emitted by the source $S(M)$ reaches the entropy $H(S(M))$ almost surely.*

5 How to build Antidictionaries

In practical applications the antidictionary is not *a priori* given but it must be derived either from the text to be compressed or from a family of texts belonging to the same source to which the text to be compressed is supposed to belong.

There exist several criteria to build efficient antidictionaries that depend on different aspects or parameters that one wishes to optimize in the compression process. Each criterium gives rise to different algorithms and implementations.

We present a simple construction to build finite antidictionaries. It is the base on which several variations are developed. It can be used to build antidictionaries for fixed sources. In this case our scheme can be considered as a compressor generator (compressor compiler). In the design of a compressor generator, or compressor compiler, statistic considerations play an important role, as discussed in Section 6 (cf. [11]).

Algorithm BUILD-AD below builds the set of minimal forbidden words of length k ($k > 0$) of the word w. It takes as input an automaton accepting the words that have the same factors of length k (or less) than w, *i.e.*, accepting the language

$$L = \{x \in \{0,1\}^* \mid (u \in F(x) \text{ and } |u| \leq k) \Rightarrow u \in F(w)\}.$$

The preprocessing of the automaton is done by the algorithm BUILD-FACT whose central operation is described by the function NEXT. The automaton is represented by both a trie and its failure function f. If p is a node of the trie associated with the word av, $v \in \{0,1\}^*$ and $a \in \{0,1\}$, $f(p)$ is the node associated with v. This is a standard technique used in the construction of suffix trees (see [7] for example). It is used here in algorithm BUILD-AD (line 4) to test the minimality of forbidden words according to the equality 2.

```
BUILD-FACT (word w ∈ {0,1}*, integer k > 0)
    i ← new state; Q ← {i};
    level(i) ← 0;
    p ← i;
    while not end of string w
        a ← next letter of w;
        p ← NEXT(p, a, k);
    return trie (Q, i, Q, δ), function f;
```

```
NEXT (state p, letter a, integer k > 0)
    if δ(p, a) defined
        return δ(p, a);
    else if level(p) = k
        return NEXT(f(p), a, k);
    else
        q ← new state; Q ← Q ∪ {q};
        level(q) ← level(p) + 1;
        δ(p, a) ← q;
        if (p = i) f(q) ← i; else f(q) ← NEXT(f(p), a, k);
        return q;
```

```
Build-AD (trie (Q, i, Q, δ), function f, integer k > 0)
T ← ∅; δ' ← δ;
for each p ∈ Q, 0 < level(p) < k, in width-first order
    for a ← 0 then 1
        if δ(p, a) undefined and δ(f(p), a) defined
            q ← new state; T ← T ∪ {q};
            δ'(p, a) ← q;
Q ← Q \ {states of Q from which no δ'-path leads to T}
return trie (Q ∪ T, i, T, δ');
```

The above construction gives rise to a *static* compression scheme in which we need to read twice the text: the first time to construct the antidictionary M and the second time to encode the text.

Informally, the encoder sends a message z of the form $(x, y, \sigma(n))$ to the decoder, where x is a description of the antidictionary M, y is the text coded according to M, as described in Section 2, and $\sigma(n)$ is the usual binary code of the length n of the text. The decoder first reconstructs from x the antidictionary and then decodes y according to the algorithm in Section 2. We can choose the length k of the longest minimal forbidden word in the antidictionary such that, by coding the trie associated to M with standard techniques, one has that $|x| = o(n)$. Since the compression rate is the size $|z|$ of z divided by the length n of the text, we have that $|z|/n = |y|/n + o(n)$. Assuming that for n and k large enough the source $S(M)$, as in Section 4, approximates the source of the text, then, by the results of Section 4, the compression rate is "optimal".

Example 1

Let $\mathbf{w}= a_1 a_2 \cdots$ be a binary infinite word that is periodic (*i.e.*, there exists integer $P > 0$ such that for any index i the letter a_i is equal to the letter a_{i+P}), and let w_n be the prefix of $\mathbf{w}$ of length n.

We want to compress the word w_n following our simple scheme informally described above. Since $\mathbf{w}$ has period P, then for any $i > P$, letter a_i is uniquely determined by the P previous letters. Therefore, we define the antidictionary

$$M = \{ua \mid u \in F(\mathbf{w}), |u| = P - 1, \text{ and } ua \notin F(\mathbf{w})\},$$

where $F(\mathbf{w})$ is the language of all factors of $\mathbf{w}$. Then it is easy to prove that for any prefix w_n of $\mathbf{w}$, the length of the text y coded using M is constantly equal to P. Hence the compression rate for w_n is $|z|/n = O(\sigma(n)) = O(log_2(n))$, which means that the method can achieve an exponential compression.

It is possible to generalize the previous example to any binary infinite word, $\mathbf{w}= a_1 a_2 \cdots$, that is ultimately periodic (*i.e.*, there exist integers $M > 0$, $P > 0$ such that for any index $i \geq M$ the letter a_i is equal to the letter a_{i+P}).

Example 2

This example is a bit more complex, and the compression rate is no more exponential in the size of the text.

We start with the classical recursive definition of finite Fibonacci words f_n (cf. [5]). Let $f_1 = \mathtt{0}$, $f_2 = \mathtt{01}$ and let $f_{n+1} = f_n f_{n-1}$ for $n \geq 2$. In particular we have $f_3 = \mathtt{010}$, $f_4 = \mathtt{01001}$ and $f_5 = \mathtt{01001010}$. The infinite Fibonacci word **f** is the limit of the sequence of the finite Fibonacci words, *i.e.*, the unique infinite word that have all the f_n as prefixes. It is know that the length $|f_n|$ is the n-th fibonacci number and, consequently, $lim\ |f_n| = \Theta(\varphi^n)$ where $\varphi \equiv 1.618\cdots$ is the golden ratio.

Let $L(\mathbf{f})$ and $L(f_n)$ be the factorial languages composed respectively by all factors of the Fibonacci infinite word **f** and of the finite Fibonacci word f_n. It is known (cf. [16]) that any factor of **f** of length $4m$ contains as factors all factors of length m of the whole word **f**. In particular $L(f_n) \cap \{\mathtt{0}, \mathtt{1}\}^m = L(\mathbf{f}) \cap \{\mathtt{0}, \mathtt{1}\}^m$ for any $m \leq |f_n|/4$. Consequently, the minimal forbidden words of **f** up to length m are also the minimal forbidden words of f_n up to length m for any $m \leq |f_n|/4$.

If we call g_n to be the prefix of length $|f_n| - 2$ of f_n for $n \geq 2$, it is known (cf. Example 2 of [4]) that all the minimal forbidden words of $L(\mathbf{f})$ are

$$\{\mathtt{1}g_{2i}\mathtt{1} \mid i \geq 1\} \cup \{\mathtt{0}g_{2i+1}\mathtt{0} \mid i \geq 1\},$$

i.e., they are 11, 000, 10101, 00100100, $\cdots$.

We now compress the word f_{2n} following our simple scheme informally described above. We choose as length k of the longest minimal forbidden word the number $k = |f_n|$. By previous observations

$$M = (\{\mathtt{1}g_{2i}\mathtt{1} \mid i \geq 1\} \cup \{\mathtt{0}g_{2i+1}\mathtt{0} \mid i \geq 1\}) \cap \{\mathtt{0}, \mathtt{1}\}^k,$$

and consequently it is not difficult to prove that the size of a standard coding x of the trie associated to M is $x = O(|f_n|) = O(\varphi^n) = o(|f_{2n}|)$.

It is possible to prove that the size of the compressed version y of f_{2n} by using our algorithm with the antidictionary M is $|y| = O(|f_n|)$.

Therefore the global compression rate is $O(|f_n|/|f_{2n}|) = O((1/\varphi)^n)$. This means that the compression ratio converges exponentially to zero as n goes up to infinity.

6 Improvements and experimental results

In the previous section we presented a static compression scheme in which we need to read twice the text. Starting from the static scheme, several variations and improvements can be proposed. These variations are all based on clever combinations of two elements that can be introduced in our model:

a) statistic considerations,

b) dynamic approaches.

These are classical features that are sometimes included in other data compression methods.

Statistic considerations are used in the construction of antidictionaries. If a forbidden word is responsible of "erasing" few bits of the text in the compression algorithm of Section 2 and its "description" as an element of the antidictionary is "expensive" then the compression rate improves if it is not included in the antidictionary. On the contrary, one can introduce in the antidictionary a word that is not forbidden but that occurs very rarely in the text. In this case, the compression algorithm will produce some "mistakes". In order to have a lossless compression, the encoder must take account of such mistakes and must also send them to the decoder. Typical "mistakes" occur in the case of antidictionaries built for fixed sources and in the dynamic approach. Even with mistakes, assuming that the mistakes are rare with respect to the longest word (length) of the antidictionary, our compression scheme preserves the synchronization property for fixed sources.

In the dynamic approach we construct the antidictionary and we encode the text at the same time. The antidictionary is constructed (also with statistical consideration) by taking account of (a part of) the text previously read.

We have realized prototypes of the compression and decompression algorithms. They also implement the dynamic version of the method. They have been tested on the Calgary Corpus, and the next table reports the size of compressed files. The total size of compressed data is equivalent to most common compressors such as pkzip.

File	compressed size
bib	35535
book1	295966
book2	214476
geo	79633
news	161004
obj1	13094
obj2	111295
paper1	21058
paper2	32282
pic	70240
progc	15736
progl	20092
progp	13988
trans	22695
Total	1,107094

7 Conclusion and Generalizations

We have described DCA, a text compression method that uses some "negative" informations about the text, which are described in terms of antidictionaries.

The advantages of the scheme are:

- fast at decompressing data,
- it is similar to a compressor generator (compressor compiler) for fixed sources,
- fast at compressing data for fixed sources,
- it has a synchronization property in the case of finite antidictionaries that leads to parallel compression and to search engines on compressed data.

We are considering several generalizations:

- compressor scheme and implementation of antidictionaries on more general alphabets,
- the use of lossy compression especially to deal with images,
- the combination of DCA with other compression scheme. For instance using both dictionaries and antidictionaries like positive and negative sets of examples as in Learning Theory,
- the design of chips dedicated to fixed sources.

Acknowledgements We thanks F.M. Dekking for useful discussions.

References

[1] A. V. Aho and M. J. Corasick. Efficient string matching: an aid to bibliographic search. *Comm. ACM* 18:6 (1975) 333–340.

[2] R. Ash. *Information Theory.* Tracts in mathematics, Interscience Publishers, J. Wiley & Sons, 1985.

[3] M. P. Béal. *Codage Symbolique.* Masson, 1993.

[4] M.-P. Béal, F. Mignosi, and A. Restivo. Minimal Forbidden Words and Symbolic Dynamics. in (*STACS'96*, C. Puech and R. Reischuk, eds., LNCS 1046, Springer, 1996) 555–566.

[5] J. Berstel. Fibonacci Words — a Survey. in (*The Book of L*, G. Rozenberg, A. Salomaa, eds., Springer Verlag, 1986).

[6] T. C. Bell, J. G. Cleary, I. H. Witten. *Text Compression.* Prentice Hall, 1990.

[7] M. Crochemore, C. Hancart. Automata for matching patterns. in (*Handbook of Formal Languages*, G. Rozenberg, A. Salomaa, eds.", Springer-Verlag", 1997, Volume 2, *Linear Modeling: Background and Application*) Chapter 9, 399–462.

[8] M. Crochemore, F. Mignosi and A. Restivo. Minimal Forbidden Words and Factor Automata. Accepted at the conference *MFCS'98*.

[9] M. Crochemore, F. Mignosi and A. Restivo. *Automata and Forbidden Words*. Technical Report, IGM 98-5, Institut Gaspard-Monge, 1998. Submitted to *Information Processing Letters*.

[10] M. Crochemore, F. Mignosi, A. Restivo and S. Salemi. Search in Compressed Data. in preparation.

[11] M. Crochemore, F. Mignosi, A. Restivo and S. Salemi. A Compressor Compiler. in preparation.

[12] R. S. Ellis. *Entropy, Large Deviations, and Statistical Mechanics.* Springer Verlag, 1985.

[13] J. Gailly. *Frequently Asked Questions in data compression*, Internet. Available at `http://www.landfield.com/faqs/compression-faq/part1/preamble.html` or `ftp://rtfm.mit.edu/pub/usenet/news.answers/compression-faq/`.

[14] J. G. Kemeny and J. L. Snell. *Finite Markov Chains.* Van Nostrand Reinhold, 1960.

[15] R. Krichevsky. *Universal Compression and Retrieval.* Kluver Academic Publishers, 1994.

[16] M. Morse and G. Hedlund. Symbolic Dynamics II: Sturmian trajectoires. *Amer. J. Math.* 62, 1-40, 1940.

[17] M. Nelson and J. Gailly. *The Data Compression Book.* 2nd edition. M&T Books, New York, NY 1996.

[18] C. Shannon. Prediction and entropy of printed english. *Bell System Technical J.*, 50-64, January, 1951.

[19] J. A. Storer. *Data Compression: Methods and Theory.* Computer Science Press, Rockville, MD, 1988.

[20] I. H. Witten, A. Moffat and T. C. Bell. *Managing Gigabytes.* Van Nostrand Reinhold, 1994.