Theoretical Computer Science 92 (1992) 213–221
Elsevier

Data compression with factor automata*

Marc Zipstein
C.E.R.I.L. 25 Cours Blaise Pascal, 91000 Evry, France

Abstract

Zipstein, M., Data compression with factor automata, Theoretical Computer Science 92 (1992) 213–221.

We present a data compression algorithm that uses the factor automaton. The method is related to the Ziv and Lempel's algorithm and gives an approximation of the entropy of a text.

Text compression algorithms are methods that reduce the number of symbols used to represent a sequence of characters, therefore reducing the amount of space needed to store it or the amount of time necessary to transmit it. Universal compression methods may be used with no a priori knowledge on the content and the structure of the text to be treated.

There are two main classes of data compression methods: block encoding and factor encoding. We present a new method of factor encoding which uses an automaton that recognises all the factors of a text.

Block encoding parses the text in blocks of a fixed length (generally the letters) and encodes each block in accordance with its probability of appearance, in such a way that a frequent letter will have a shorter translation than a less frequent one. The best-known block encoding method is the Huffman encoding [4] which uses a prefix code of minimal average length.

Factor encoding parses the text in words of different lengths, and uses a dictionary to translate each word. This dictionary may be fixed or it may be constructed as the text is processed.

The most famous factor encoding algorithm is the Ziv and Lempel's encoding [9] which uses a dictionary closed by prefix. Ziv and Lempel have proposed another

*Work supported by P.R.C. Mathematics and Computer Science.

factor encoding, where any factor of the text previously treated may be used. In their article [5], they do not give any method for the determination of those factors. Rodeh et al. [6] described an implementation of this encoding based on the factor tree. Our method, described in Section 1, uses the factor automaton structure which gives a compact representation of the set of the factors of the text. It is constructed dynamically as the text is processed. The construction of the automaton and its use for encoding is given in Section 2. The method gives a better compression ratio than the Ziv and Lempel's method on texts with low entropy and, as shown in Section 3, it also gives an approximation of the entropy of the text; this approximation converges towards the real value quicker than the one obtained by the Ziv and Lempel's method.

1. The factor automaton and the factor encoding

We consider a finite alphabet A. Let A^* be the set of the words on the alphabet A. We denote by $|w|$ the length of the word w.

The set of factors of a text T is:

$$F(T)=\{u\in A^*:\ \exists v,w\in A^*\ vuw=T\}.$$

We denote the factor automaton of the text T by $\mathscr{F}(T)$. The construction of this automaton is deduced from the one of suffix automaton, first presented by Blumer et al. [1]. We use another construction of the factor automaton due to Crochemore [2].

We define the function *position* on the states of $\mathscr{F}(T)$ by: *position*(q) is the position of the first occurrence of the longest word w that leads from the initial state to the state q. This function is well defined because one property of the factor automaton defined above is that there is a single word w_q of maximum length that leads from the initial state to a given state q, and all the words leading to state q are suffixes of w_q.

We assign the position of the first letter of the text to be one, in order to preserve the value 0 to indicate that a letter has not yet appeared in the text.

The automaton is built as the text is processed.

The encoding of a text T is defined by induction on prefixes of increasing length of T: let t be a prefix of T. We assume the automaton $\mathscr{F}(t)$ has been constructed, and that t has been encoded.

If the letter following t has never appeared in t then it is encoded by the pair (0, ASCII code of the letter). Otherwise, the next word to be translated is $w=w_1w_2\dots w_n$, $w_i\in A$, such that tw is a prefix of T and w is the longest word such that for every i, $1\leqslant i\leqslant n$, $w_1w_2\dots w_i$ is a factor of $tw_1\dots w_{i-1}$. The function *position* gives the position p of the first occurrence of w. The word w is encoded by two integers: its position p and its length $|w|$.

Any sequence of pairs, corresponding to the encoding of a text, can be deciphered. The reason is that the decipherer and the encipherer have the same automaton when they treat the same part of the text. To treat $(p,|w|)$, the decipherer just has to follow the transitions of the automaton, on the path that represents the text, from the state

p during $|w|$ letters. Those letters are the ones of w, and the automaton dynamically changes from $\mathscr{F}(t)$ to $\mathscr{F}(tw)$.

Example.
Encoding of aabbabbab

Text	*a*	*a*	*b*	*b*	*abbab*
Translation	0*a*	1 1	0*b*	3 1	2 5

Decoding of 0*a* 1 1 0*b* 3 1 2 5

Assume 0*a* 1 1 0*b* 3 1 has been treated. The factor automaton for *aabb* has been constructed (see Fig. 1).

To translate the pair 2 5 the algorithm starts from state 2, corresponding to the position of the encoded factor in the text, and follows five transitions along the main path (corresponding to the text). The automaton is completed as the translation is made; so, it is possible to follow five transitions from state 2. During the decoding of the pair 2 5, after the translation of three letters, the automaton becomes as shown in Fig. 2.

The remaining problem is an implementation problem. An efficient coding of the integers has to be used because translating a factor needs two integers whereas the Ziv and Lempel encoding only needs one. On the other hand, the translation of the whole text needs less factors because Ziv and Lempel's algorithm does not use every possible factor, and our algorithm does.

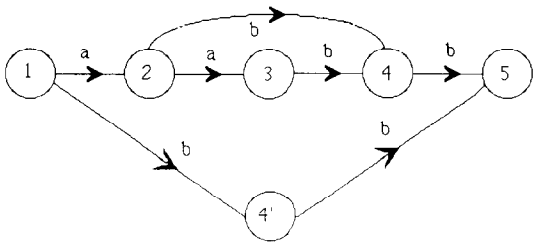

Fig. 1.

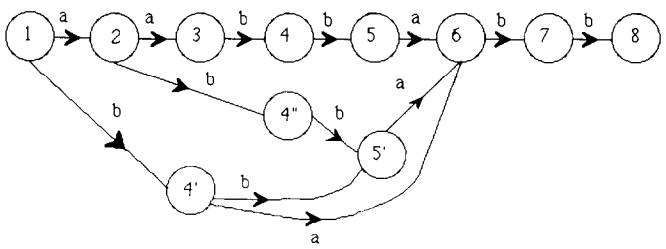

Fig. 2.

2. Construction of factor automaton and encoding

In order to save memory space, the implementation of the automaton uses a failure function, defined below. Such a failure function is used by Knuth, Morris and Pratt in their algorithm for pattern-matching in strings.

Failure function

Let $\mathscr{A}=(Q, A, i, T, d)$ be an automaton, d its transition function. Let t be a function from $Q \times A$ to Q (t is a transition function) and s a function from Q into itself.

We say that the pair (t, s) represents the transition function d if both the following conditions hold: (1) $t \subset d$, (2) $d(q, a)=d(s(q), a)$ whenever $t(q, a)$ is not defined while $d(q, a)$ and $s(q)$ are. ($s(q)$ is a stand-in of state q.) The function s is called a failure function.

Construction of the factor automaton for a text T

The algorithm given below simultaneously constructs the transition function, the failure function of the factor automaton and determines the pair that translates a factor.

It makes use of a function "*length*" defined on the states of the automaton by

$$length(q)=\max\{|u|,\ u \in A^* \text{ and } d(i, u)=q\}.$$

Coding

To determine the pair that translates a factor, the algorithm keeps an index, Index, that follows the transitions as the letters of T are read. If no transition from the current state is allowed for the current letter, then a new coding pair has been found, and the index goes back to the initial state (see Fig. 3).

Remarks

The number of states in the automaton is at most twice the length of the text T since at most two new states may be created in the main **While**.

Each state of the automaton is a final state.

We use the term "*factor automaton*" despite the fact that *factor automaton* usually denotes the minimal automaton that recognises the factors of a text.

3. Parsing with the method of factor and entropy

The notion of entropy for a text T was first introduced by Shannon [7]. Entropy measures the quantity of information contained by a text, and corresponds to the minimal average number of symbols necessary to write a letter of the text.

```
Factor_Automaton(T)
     {
     Create_a_state(Initial);Length(Initial) = 0;
     Last = Initial;Index=Initial;
     Length_w=0;Position(Initial)=1;
     While (the end of T is not reached)
          {
          Create_a_state(q);
          a = (next letter of T);
          p = Last;Position(q)=Position(Last)+1;
/*Encoding*/
          If (d(Index, a) undefined)
               If (Index=Initial) Send(0,ASCII(a));
               Else      {
                    Send(Position(Index)-Length_w,Length_w)
                    Index=Initial;Length_w=0;
                    }
          Else      {
               Index=d(Index, a);Length_w =Length_w+1;}
/*Up-date of the automaton*/
          While (p ≠ initial and d(p, a) undefine)
               {d(p, a) = q;p = s(p);}
          If (d(p, a) undefine)
               {d(Initial, a) = q;s(q) = Initial;}
          Else
               If (Length(p)+1 = Length(d(p, a)) )
                    s(q) = d(p, a);
               Else {
                    Create_a_state(r,copy of state d(p, a));
/* same transitions, same stand-in state,same position*/
                    Length(r) = Length(p)+1;
                    s(d(p, a)) = r; s(q) = r;
                    While (Length(d(p,a)) ≥ Length(r))
                              {d(p, a) = r; p = s(p) ;}
                    }
          }
     }
```

Fig. 3. The algorithm for the construction of the factor automaton of a text T.

If the statistics of the text T are known, the probability of appearance of letter a being $p(a)$ the entropy $H(T)$ is

$$H(T)=-\sum_{a\in A} p(a)\log p(a).$$

When the probability of the source is unknown, Ziv and Lempel's method gives an asymptotical approximation of the entropy. This approximation is based on the number of factors used during the translation of the text.

Definition. Let $T\in A^*$, $T=(a_n)$, $n\in N$. The Z-factorisation of T is the sequence of words $(w_n)=(w_n(T))$ defined by induction: $w_0=a_0$. If $w_0, w_1, \ldots, w_k$ are defined and $w_0 w_1 \ldots w_k = a_0 a_1 \ldots a_n$ then $w_{k+1}=a_{n+1}a_{n+2}\ldots a_{n+m}$, where m is the least integer such as $a_{n+1}a_{n+2}\ldots a_{n+m}\notin\{w_0, w_1, \ldots, w_k\}$.

For every $T\in A^*$, we define $n_Z(k, T)=|w_0(T)w_1(T)\ldots w_k(T)|$, i.e. the length of the text reached with the k first words of the Z-factorisation.

The value $k\log k$ represents the number of bits necessary to write k factors and $k\log k/n_Z(k, T)$ is the average number of bits necessary to write a letter of T.

So,

$$H(T)\leqslant\frac{k\log k}{n_Z(k, T)}.$$

It has been shown [9, 3] that for almost every T, the entropy H is

$$H(T)=\lim_{k\to\infty}\frac{k\log k}{n_Z(k, T)}.$$

The parsing with the factors also gives an approximation of the entropy, and this approximation converges rapidly.

Definition. The F-factorisation of T is the sequence of words $(v_n)=(v_n(T))$ defined by induction: $v_0=a_0$. If $v_0, v_1, \ldots, v_k$ are defined and $v_0 v_1 \ldots v_k = a_0 a_1 \ldots a_n$ then v_{k+1} is a new letter or $v_{k+1}=a_{n+1}a_{n+2}\ldots a_{n+m}$, where m is the least integer such as $a_{n+1}a_{n+2}\ldots a_{n+m}$ is not a factor of $v_0 v_1 \ldots v_k$.

F-factorisation was introduced in [5] and used in [2].

We define $n_F(k, T)=|v_0(T)v_1(T)\ldots v_k(T)|$.

Proposition. *Let T be an infinite text. For every integer k, the k first words of the Z-factorisation cover a shorter prefix of text than the k first words of the F-factorisation:* $n_Z(k, T)\leqslant n_F(k, T)$.

The F-factorisation always chooses the longest possible factor and all the words used by the Ziv and Lempel's algorithm are factors.

The value $k \log k / n_F(k, T)$ is the average number of bits necessary to write a letter of T.

$$H(T) \leqslant \frac{k \log k}{n_F(k, T)}.$$

So,

$$H(T) \leqslant \frac{k \log k}{n_F(k, T)} \leqslant \frac{k \log k}{n_Z(k, T)}$$

and

$$H(T) = \lim_{k \to \infty} \frac{k \log k}{n_F(k, T)}.$$

An estimation of the entropy is given from the parsing of a text with its factors.

Experimental approximation entropy

We construct the programs texts where the probability of appearance of each letter is known; so, a direct computation of the entropy is feasible. We use an alphabet with 20 letters.

We compare the values $k \log k / n_F(k, T)$ and $k \log k / n_Z(k, T)$ with the entropy while more text was treated.

The entropy of the source is the value calculated with the exact number of occurrences of the letters in the text.

(1) *Equiprobability*: Every letter of the text has the same probability of appearance $p(a_i) = 0.05$, $0 \leqslant i < 20$. The theoretical entropy is 4.321929.

(2) All the letters but one have the same probability of appearance $p(a_i) = 0.01$, $0 < i < 20$. The remaining letter a_0 has a probability of 0.81. The theoretical entropy is 1.508577.

The estimation obtained by the method of factors converges to the exact value of the entropy faster than the one obtained by Ziv and Lempel's method (see Tables 1 and 2).

Table 1

Length	Entropy of the source	Approximation with factors	Approximation by Ziv and Lempel [9]
5000	4.319277	5.237993	5.794274
10000	4.320462	5.099692	5.745548
20000	4.321409	4.93616	5.631763
30000	4.321482	4.857694	5.559893
40000	4.321614	4.818317	5.502391
50000	4.321640	4.797632	5.458225

Table 2

Length	Entropy of the source	Approximation with factors	Approximation by Ziv and Lempel [9]
5000	1.554181	1.768732	2.252765
10000	1.550892	1.774483	2.205569
20000	1.515825	1.682011	2.106605
30000	1.524914	1.656781	2.086431
40000	1.508161	1.608550	2.039223
50000	1.503307	1.580984	2.016964
60000	1.497791	1.560479	1.998559
70000	1.501193	1.552968	1.991666

Table 3

Source	French	C	Lisp	Equiprob	Alphabet
Length	62816	684497	75925	70000	530000
Fact	29795	**218057**	**26944**	51617	**481**
	47.43%	**31.86%**	**35.49%**	73.74%	**0.09%**
Compress	**26043**	233855	30767	288699	11309
	41.46%	34.16%	40.52%	63.60%	2.13%
Huffman	33462	425063	48332	**38909**	385067
	53.27%	62.10%	63.66%	**55.58%**	72.65%

4. Experimental results

We have compared (see Table 3) the performances of the various data compression algorithms: the one based on the factor automaton, the Unix command "*compress*" which is an implementation of the variation of the Ziv and Lempel's method proposed by Welch [8], and the Huffman coding implemented on Unix as the command "*pack*".

We tried different kinds of texts: texts in French, programs written in C or Lisp, and specially built texts: "Equiprob" is a text written on an alphabet of twenty letters, each letter having the same probability of appearance; "Alphabet" is the repetition of the line "$ab \ldots zAB \ldots Z$".

In our implementation, named "Fact", the automaton is discarded and recreated from scratch when it has 125 000 states.

The method that uses the factor automaton is more efficient on texts with low entropy where it can find long factors, as in source programs or "alphabet".

On texts with higher entropy, as texts in natural language, frequent long factors are rare and the use of two integers to code one factor leads to a poorer performance than the Ziv and Lempel's algorithm.

As the entropy increases, the block encoding methods, as the Huffman encoding, are more efficient.

References

[1] A. Blumer, J. Blumer, A. Ehrenfeucht, D. Haussler, M.T. Chen and J. Seiferas, The smallest automaton recognizing the subwords of a text, *Theoret Comput. Sci.* **40**(1) (1985) 31–56.

[2] M. Crochemore, Transducers and repetitions, *Theoret. Comput. Sci.* **45** (1986) 63–86.

[3] G. Hansel, Estimation of the entropy by the Lempel–Ziv method, in: Gross and Perrin, eds., *Electronic Dictionaries and Automata in Computational Linguistics* (actes de l'école de printemps, Oléron, 1987) Lecture Notes in Computer Science (Springer, Berlin, 1989) 51–65.

[4] D.A. Huffman, A method for the construction of minimum redundancy codes, *Proc. IRE* **40** (1952) 1098–1101.

[5] A. Lempel and J. Ziv, On the complexity of finite sequences, *IEEE Trans. Inform. Theory* **IT 22**(1) (1976) 75–81.

[6] M. Rodeh, V.R. Pratt and S. Even, Linear algorithm for data compression via string-matching, *J. ACM* **28** (1981) 16–24.

[7] C. Shannon, A mathematical theory of communication, *Bell System Tech. J.* **27** (1948) 379–423 and 623–656.

[8] T.A. Welch, A technique for high-performance data compression, *IEEE Comput.* **17**(6) (1984) 8–19.

[9] J. Ziv and A. Lempel, Compression of individual sequences via variable length-coding, *IEEE Trans. Inform. Theory* **IT 24** (1978) 530–536.