ELSEVIER

Discrete Applied Mathematics 64 (1996) 31–55

DISCRETE
APPLIED
MATHEMATICS

Robust universal complete codes for transmission and compression

Aviezri S. Fraenkel[a], Shmuel T. Klein[b,*]

[a]Department of Applied Mathematics and Computer Science, The Weizmann Institute of Science, Rehovot 76100, Israel

[b]Department of Mathematics and Computer Science, Bar Ilan University, Ramat Gan 52900, Israel

Received 15 June 1992; revised 7 June 1994

Abstract

Several measures are defined and investigated, which allow the comparison of codes as to their robustness against errors. Then new universal and complete sequences of variable-length codewords are proposed, based on representing the integers in a binary Fibonacci numeration system. Each sequence is constant and needs not be generated for every probability distribution. These codes can be used as alternatives to Huffman codes when the optimal compression of the latter is not required, and simplicity, faster processing and robustness are preferred. The codes are compared on several "real-life" examples.

1. Motivation and introduction

Let $A = \{A_1, A_2, \ldots, A_n\}$ be a finite set of *elements*, called *cleartext* elements, to be encoded by a static uniquely decipherable (UD) code. For notational ease, we use the term "code" as abbreviation for "set of codewords"; the corresponding encoding and decoding algorithms are always either given or clear from the context. A code is *static* if the mapping from the set of cleartext elements to the code is fixed during the encoding of the text [23]. In this paper we restrict attention to static codes, thus excluding adaptive methods [26], and in particular the popular LZ techniques [28, 29]. Let p_i be the probability of occurrence of the element A_i. The elements can be single characters, pairs, triplets or any m-gram of characters, they can represent words of a natural language, they can finally form a set of items of a completely different nature, provided that there is an unambiguous way to decompose a file into a sequence of these items, in such a way that the file can be reconstructed from this sequence (see, for example, [12]). We thus think also of applications where n, the size

*Corresponding author.

SSDI 0166-218X(93)00116-2

of A, can be large relative to the size of a standard alphabet. Several criteria may govern the choice of a code. We shall concentrate on the following: (i) robustness against errors, (ii) simplicity of the encoding and decoding process, and (iii) compression efficiency.

If l_i is the length in bits of the binary codeword chosen to represent A_i, it is well known that the weighted average length of a codeword, $\sum p_i l_i$, is minimized using Huffman's [18] procedure. However, Huffman codes are extremely error sensitive: a single wrong bit may render the tail of the encoded message following the error useless. As to (ii), a new set of codewords must be generated for each probability distribution, and the encoding and decoding algorithms are rather involved.

One approach to limit the possible damage of errors is to add some redundant bits which can be used for error detection or even correction. This obviously diminishes compression efficiency and complicates further the coding procedures.

The simplest possible codes are fixed-length codes, which can be considered as robust, since an error inverting a single bit causes the loss of only one codeword. But from the compression point of view, static fixed-length codes (both fixed-to-fixed and variable-to-fixed length codes) are optimal only if the probability distribution of the cleartext elements is uniform or almost uniform, and can be very wasteful for other probability distributions. Moreover, if a bit is lost or an extraneous bit is picked up, this causes a shift of the remaining tail, which is thus lost.

The compression capabilities of codes are compared by means of their weighted average codeword lengths, and the simplicity of the coding and decoding procedures can be measured by the time and space complexity of their algorithms. In the next section, we define a *sensitivity factor*, which enables a quantitative comparison of codes regarding their robustness against errors. We then review some codes appearing in the literature and evaluate their sensitivity factor. Some classes of infinite codes are considered in Section 3 as to the simplicity of their coding algorithms and to their compression efficiency. In Section 4, a new family of variable-length codes is introduced, which can be considered as a compromise between Huffman and fixed-length codes with respect to the three above-mentioned criteria. The new family of codes depends only on the number of items to be encoded and the ordering of their frequencies, not on their exact distribution, and is based on the binary Fibonacci numeration system (see [27]). The corresponding coding algorithms are very simple. Our paper is related to [1], where various representations of the integers, based on Fibonacci numbers of order $m \geqslant 2$, are investigated, with an application to the transmission of unbounded strings. In the present work we assume an underlying probability distribution and explore the properties of Fibonacci representations for variable-length codeword sets, in particular the trade-off between their robustness and their compression efficiency. In Section 5, the codes are compared numerically on various probability distributions of "real-life" alphabets.

The broad area of data compression has been ably reviewed in [25, 23], and more recently in [26, 2]; thus we refrain from giving a review here, and cite only those works connected to the present investigation.

Throughout we restrict ourselves to binary codes, though all the ideas can be generalized to arbitrary base $\geqslant 2$. In particular, the binary codes based on the binary Fibonacci numeration system may be generalized to codes based on the sequence of integers $\{a_1^{(m)}, a_2^{(m)}, \ldots\}$, defined by $a_0^{(m)} = a_1^{(m)} = 1$ and the recurrence relation $a_i^{(m)} = ma_{i-1}^{(m)} + a_{i-2}^{(m)}$ for $i > 1$, for any fixed positive integer m ($m = 1$ is the Fibonacci case). The resulting codes are $(m + 1)$-ary codes, and their properties have been investigated by Fraenkel [10, 11].

2. Robustness

When reliable transmission of a message is needed, error-correcting codes may be used. Often, however, we do not care about single (e.g., transmission or typing) errors, as long as their influence remains locally restricted. We need a measure which enables us to compare codes according to their error-sensitivity.

2.1. *The sensitivity factor*

Let $\mathscr{F}$ be a family of errors that may occur in an encoded string, e.g., deletion or complementation of a bit, etc. Intuitively, we would consider a code C more robust than another code D, if, given any error from $\mathscr{F}$ in $S(C)$ (the string encoded by C) and any error from $\mathscr{F}$ in $S(D)$, the number of misinterpreted codewords for C is smaller than for D. Henceforth, we restrict $\mathscr{F}$ to contain substitution, as well as deletion and insertion errors. That is, "an error occurring at position x" has to be understood as either x changing its value v to $1 - v$, or x being lost, or that a 0 or 1 bit was inserted just to the right of x.

We propose as measure the "expected maximum" number of codewords which may be lost when a single error occurs; the expected maximum is obtained by calculating the maximum for all the possible locations of the error and then averaging appropriately. More formally, let C be a code with codewords c_i of length l_i which appear with probability p_i, $1 \leqslant i \leqslant n$; let q_i be the probability that a bit at a randomly chosen location of a long encoded string belongs to c_i. Note that q_i is proportional to both p_i and l_i, that is, $q_i = p_i l_i / \sum_{j=1}^{n} p_j l_j$; in particular, for fixed length codes, $q_i = p_i$. Let $M(c_i, j)$ be the maximal number of codewords which may be lost if an error occurs in the jth bit of c_i, $1 \leqslant j \leqslant l_i$. Assuming that any bit in c_i has equal chance to be erroneous, let $M(c_i) = (1/l_i)\sum_{j=1}^{l_i} M(c_i, j)$ be the expected maximum number of codewords which may be lost if an error occurs in c_i. The *sensitivity factor* of C is defined as

$$\mathscr{S}\mathscr{F}(C) \overset{\text{def}}{=} \sum_{i=1}^{n} q_i M(c_i) = \frac{1}{L} \sum_{i=1}^{n} p_i \sum_{j=1}^{l_i} M(c_i, j), \tag{1}$$

where $L = \sum_{j=1}^{n} p_j l_j$ is the average codeword length.

The reason for preferring the "expected maximum" over the "expected average" in the definition of $\mathscr{SF}$ is a technical one: the average number of codewords lost by an error in a given bit depends on the entire set of codewords and their distribution, and is thus often much harder to evaluate than the maximum, which is independent of the distribution. We now evaluate the sensitivity of several known codes, which we consider in order of increasing $\mathscr{SF}$.

An absolutely robust code T would be, e.g., a code with a representation of each codeword by a triple replication of itself: transmit every bit three times and retain the value which occurred at least twice. Under our assumption of a *single* error, no codeword would be misunderstood, thus $\mathscr{SF}(T) = 0$. But there are more economical error correcting codes if such low sensitivity is required.

In order to get better compression, variable-length codes should be used. These are on the one hand more vulnerable than fixed-length codes, because even a substitution error can change a codeword into one of different length, and the error can thus propagate. On the other hand, an insertion or deletion error will cause more damage to a fixed-length code F, for which synchronization will be lost "forever", i.e., $\mathscr{SF}(F)$ is unbounded, whereas certain variable-length codes might resynchronize sooner or later. For a finite set of cleartext elements, optimum compression is obtained by Huffman codes, but as was already mentioned, they have to be generated for each probability distribution. We first consider some fixed infinite sets of variable-length codewords, which yield inferior compression but are much easier to use, as any set of n elements is now encoded by the following simple procedure:

1. Sort the probabilities into non-increasing order: $p_1 \geqslant \cdots \geqslant p_n$.
2. Assign the ith codeword (which were sorted by non-decreasing length) to the element whose probability is p_i.

The encoding and decoding algorithms are then simply based on table lookups.

The simplest variable-length code is a *unary* code $\mathscr{U} = \{1, 01, 001, 0001, \ldots\}$, i.e., the ith codeword consists of a 1 preceded by $i - 1$ zeros, $i = 1, 2, \ldots$. Such a code should be used only for distributions which are close to $p_i = 2^{-i}$. If an error deletes the 1 at the right end of any codeword or changes it into a zero, then two adjacent codewords fuse together, so there are two misinterpretations. An insertion of 0 at the last bit only affects the following codeword, while an insertion of 1 at the last bit just adds a new codeword. If an error occurs elsewhere (in one of the zeros), the current codeword will be decoded as if there were two codewords (in case of substitution or insertion of 1), but only one codeword is lost. For $\mathscr{U}_n$, the first n codewords of $\mathscr{U}$, the average codeword length is $L = \sum_{i=1}^{n} ip_i$, thus we get from (1),

$$\mathscr{SF}(\mathscr{U}_n) = \frac{1}{L}\sum_{i=1}^{n} p_i((i-1)+2) = 1 + \frac{1}{L}.$$

In [13], the following method for generating *block*-codes of length N is proposed. These are also called *prefix-synchronized codes* [14], which are special cases of *comma free* codes (see, e.g., [20]): fix any binary pattern π of $k < N$ bits and consider the set of all strings of the form $y = \pi x$, where x is a binary string of length $N - k$ such that the pattern π occurs in $\pi x \pi$ only as prefix and suffix. This allows the receiver of an encoded message to resynchronize (e.g., after a transmission error) by looking for the next appearance of the pattern π.

Another variant appears in [22], who studied variable-length codes. As he did not consider the above synchronization problem, but was interested mainly in UD codes, he defined the set of strings of the form $y = x\pi$ (now π occurs as *suffix* in every codeword), where π is as above and x is a binary string of length at least 1 bit, but the restriction on x being only that π occurs in y exactly once and as suffix. Hence one obtains a prefix-code. The set of all binary strings of length $\geqslant k$ in which π occurs only as suffix is called the set *generated* by π, and will be denoted $\mathscr{L}(\pi)$. Note that we have adjoined π itself to the code defined by Lakshmanan, in order to get better compression. In [3], $\mathscr{L}(\pi)$ is called a *semaphore* code.

Various choices of π are investigated in [13]. Gilbert conjectured that the number $G(N)$ of possible codewords of length N can be maximized by choosing a prefix of the form $\pi = 1 \ldots 10$ of suitable length k. This conjecture was proved by Guibas and Odlyzko [14] for large N, who showed, more generally, that $G(N)$ is maximized by the choice of a prefix π with *autocorrelation* $10 \ldots 0$ ($k - 1$ zeros) and with length k such that $|k - \log_2 N| \leqslant 1$. A binary string x has autocorrelation $10 \ldots 0$ if and only if no proper prefix of x is identical to any suffix of x. For example, $x = 1 \ldots 10$ has autocorrelation $10 \ldots 0$.

Suppose π has autocorrelation $10 \ldots 0$. If an error occurs in a codeword x of length ℓ in one of its $\ell - k$ leftmost bits (not in π), then only x is lost. Indeed, inserting, deleting or changing a bit can either cause the prefix to be altered, or it can create a new occurrence of the pattern π. However, in the latter case, the new occurrence of π cannot have overlapping bits with the suffix π of x, because π has autocorrelation $10 \ldots 0$. Thus the altered codeword x will possibly be decoded as two codewords, but the following codewords are not affected.

If an error occurs in one of the bits of the suffix π of x, then a new occurrence of π can be created which has overlapping bits with the suffix π of x, even if π has autocorrelation $10 \ldots 0$. An example of such a pattern is $\pi = 1110110$; if the codeword is $x = 11110\pi$, it would be decoded after a substitution error in the third bit from the left in π as $1\pi0110$; if the codeword following x is $y = 01100\pi$, then a substitution error in the rightmost bit of x would yield the decoding $111101110\pi0\pi$. If there is no occurrence of π in the concatenation of the altered form of x with the prefix of y, then only x and y are lost, and if there is a new occurrence of π, it cannot be partly overlapping with the suffix π of y; hence in any case, only two codewords are lost. Therefore we get from (1)

$$\mathscr{S}\mathscr{F}(\mathscr{L}_n(\pi)) = \frac{1}{L}\sum_{i=1}^{n} p_i((l_i - k) + 2k) = 1 + \frac{k}{L},$$

where $\mathscr{L}_n(\pi)$ is the set of the n first elements of $\mathscr{L}(\pi)$, ordered by non-decreasing codeword length. The above unary code is the special case $\pi = 1$.

Suppose now that π has autocorrelation other than $10\ldots0$. Then $\mathscr{S}\mathscr{F}$ is not necessarily bounded. Consider for example the pattern $\pi = 1110011$, and the following encoded message, in which occurrences of π are overlined:

$$\ldots 111\overline{1110011}100\overline{1110011}11100\overline{1110011}1100\overline{1110011}1 \ldots ;$$

a substitution error in the leftmost bit of the leftmost occurrence of π would yield the decoding:

$$\ldots 11101100\overline{1110011}10011\overline{1110011}1001\overline{1110011}100111 \ldots ,$$

and this example can be extended arbitrarily. Thus $\mathscr{S}\mathscr{F}(\mathscr{L}_n(\pi))$ is not bounded, when the number of codewords tends to infinity.

In [6], a code $\mathscr{R} = \{r_1, r_2, \ldots\}$ is proposed which encodes the cleartext element A_i by a *logarithmic ramp* representation of the integer i. The first element r_1 is 0. Let $B(x)$ denote the standard binary representation (with leading 1) of the integer x. Then for $i > 1$, r_i is obtained in the following way: $B(i)$ is prefixed by $B(\lfloor \log_2 i \rfloor)$, and the process of recursively placing the length of a string (minus 1) in front of that string is repeated until a string of length 2 is obtained. Since all the strings $B(x)$ have a leading 1-bit, the bit 0 is used to mark the end of the logarithmic ramp. For example, r_{16} is 10-100-10000-0 and r_{35} is 10-101-100011-0, where dashes have been added for clarity. A substitution error in one of the $\lceil \log_2 i \rceil + 1$ rightmost bits of r_i (except for the appended zero) does not change its length, so it is the only codeword to be lost. However, an insertion or deletion error, as well as any error in one of the other bits may change the codeword into one of different length, so that decoding of the following codeword does not start where it should, and such an error can propagate indefinitely, so that $\mathscr{S}\mathscr{F}(\mathscr{R})$ is not bounded. The same result holds for a similar logarithmic ramp code discussed in [7].

Finally, for a Huffman code H, an error may be self-correcting after a few codewords, even if it is not a fixed-length code (see [4]). Nevertheless, it is easy to construct arbitrarily long sequences of codewords which are scrambled by a single error, so that $\mathscr{S}\mathscr{F}(H)$ is not bounded, when the number of encoded cleartext elements grows indefinitely.

2.2. Sensitivity of synchronous codes

For the last examples, where $\mathscr{S}\mathscr{F}$ is not bounded, a more delicate definition of $\mathscr{S}\mathscr{F}$ might be used to respond to our intuitive notion of robustness, since even among those error-sensitive codes, there are some which are more robust than others. For instance, in [8], a method is proposed for certain classes of probability distributions, yielding Huffman codes which are self-synchronizing in a probabilistic sense: each code contains a so-called *synchronizing* codeword c, such that if c appears in the encoded string, the codewords following it are recognized, regardless of possible errors preceding c. More formally, a codeword $s = s_1 \ldots s_m$ is defined in [8] to be synchronizing if it satisfies the following conditions:

1. for any other codeword x, s does not appear as substring in x, except possibly as suffix;
2. if a proper prefix $s_1 \ldots s_j$ of s is a suffix of some codeword, the corresponding suffix $s_{j+1} \ldots s_m$ of s is a string of codewords.

Hence the existence of a synchronizing codeword bounds the expected length of the propagation of an error without increasing the redundancy of the code, but the authors show that there are distributions for which no such synchronous Huffman code can be constructed. In our definition of sensitivity, the existence, for certain codes, of synchronizing codewords should be taken into account.

Define for any code $C = \{c_1, \ldots, c_n\}$, the sensitivity factor $\mathcal{SF}'(C)$ similarly to $\mathcal{SF}$, as

$$\mathcal{SF}'(C) \overset{\text{def}}{=} \sum_{i=1}^{n} q_i \frac{1}{l_i} \sum_{j=1}^{l_i} S(c_i, j).$$

Here $S(c_i, j)$ is defined as the expected number of codewords between c_i and the following synchronizing codeword s (including s, but not c_i), in case the error in the jth bit changed c_i into a codeword of different length; otherwise, if the error in the jth bit changed c_i into another codeword of the same length, only c_i is lost, so define $S(c_i, j) = 1$. Note that $S(c_i, j)$ is not the expected number of codewords lost, $E(c_i, j)$, since there are possibly codewords which recover from certain errors in some c_i, but the definition of a synchronizing codeword requires it to resynchronize after *every* possible error, hence $S(c_i, j) \geqslant E(c_i, j)$. On the other hand, $S(c_i, j) \leqslant M(c_i, j)$ so that $\mathcal{SF}'(C) \leqslant \mathcal{SF}(C)$; therefore $\mathcal{SF}'(C) > \mathcal{SF}(D)$ shows that D is more robust than C for both definitions of the sensitivity factor.

The evaluation of $\mathcal{SF}'(C)$ is easy: if q is the sum of the probabilities of the synchronizing codewords in C, then $S(c_i, j)$ is either 1 or $1/q$, so that $\sum_{j=1}^{l_i} S(c_i, j) = t_i + (l_i - t_i)/q$, where t_i is the number of possibilities to transform c_i into a codeword of the same length by changing a single bit. It should be noted that there are codes which have no synchronizing codeword, but still have synchronizing *sequences*. We preferred however not to take this into account for the definition of the $\mathcal{SF}'$.

For $\mathcal{L}_n(\pi)$, where π is of length k bits, at least any codeword $x = x_1 \ldots x_\ell$ with length $\ell \geqslant 2k - 1$ is synchronizing. Condition 1 above is obviously satisfied for every codeword in $\mathcal{L}(\pi)$. As to condition 2, all the suffixes of length $\geqslant k$ are themselves codewords. The suffixes of length $< k$ of x need not to be checked: the corresponding prefixes of x have length $\geqslant k$, so if any such prefix is the suffix of a codeword, its rightmost bits are π, but this contradicts the fact that $x \in \mathcal{L}(\pi)$. In particular, every codeword of the unary code $\mathcal{U}$ is synchronizing. Thus $\mathcal{SF}'(\mathcal{L}_n(\pi)) \leqslant 1/\sum_{\{j: l_j \geqslant 2k-1\}} p_j$, and $\mathcal{SF}'(\mathcal{U}_n) = 1$.

If we consider Elias' infinite code $\mathcal{R}$, it is certainly not synchronous. The codeword r_i, for $i > 1$, can be regarded as the standard binary representation of some integer

$j > i$, thus r_i appears as substring in r_j, where it is followed by 0, violating the first condition. However, for finite codes $\mathcal{R}_n = \{r_1, \dots, r_n\}$, synchronizing codewords can be found in certain cases. For example, if $16 \leqslant n < 32$, then r_{16} is synchronizing: it is of maximal length, so the first condition is trivially satisfied, and every suffix of r_{16} is a sequence of codewords.

Since it is not always possible to construct a synchronous Huffman code, there are certain cases for which even $\mathcal{SF}'(H)$ will not be bounded. For all the examples in Section 5, synchronous Huffman codes are chosen, and their sensitivity factor $\mathcal{SF}'$ is compared with $\mathcal{SF}$ of the other codes.

2.3. A robustness vs. compression trade-off for Huffman codes

The high error-sensitivity of Huffman codes suggests that for certain applications it may be profitable to improve $\mathcal{SF}$ at the cost of a reduced compression efficiency. When only substitution errors are possible, this can be achieved by grouping the codewords in blocks of fixed size m; if the last bit of the block is not the last bit of a codeword, i.e. there is a codeword w, the tail of which does not fit into the block, then w in its entirety is moved to the beginning of the next block. In order to avoid incorrect interpretations, the last bits of the first block remain unchanged, i.e. they contain a prefix of w. As a consequence, the average length of a codeword will increase.

A Huffman coded message is deciphered by repeated traversals of the corresponding Huffman tree. Starting at the root, one passes from one level to the next lower one following the left (resp. right) pointer, if the next bit of the input string is 0 (resp. 1), until a leaf is reached; this leaf corresponds to a codeword, which is output, and the algorithm proceeds again from the root. Using m-bit blocks, the decoding procedure has to be modified as follows: every time the pointer P which points to the current place in the Huffman tree is updated, i.e. when passing to a left or right son or – when a leaf was reached – resetting the pointer to the root, a counter CN is incremented. When $\mathrm{CN} = m$, this indicates that we have completed the processing of an m-bit block, so P is set to point to the root, regardless of whether a leaf was reached or not, and the counter is zeroed. Therefore, a possible substitution error cannot affect neighboring m-bit blocks. Insertion and deletion errors however have the same devastating effect as for fixed-length codes.

We thus consider in this subsection only substitution errors, as is done for example in [15], and define a new sensitivity factor $\mathcal{SF}''$ similar to $\mathcal{SF}$, but with this restricted interpretation of the word "error". Clearly, $\mathcal{SF}''(C) \leqslant \mathcal{SF}(C)$ for any code C.

The parameter m can often be chosen so as to obtain a predetermined $\mathcal{SF}''$ or average codeword length, but obviously must not be smaller than the maximal codeword length. One can always choose the block size m to be relatively prime to the greatest common divisor of all the codeword lengths, and then one can assume that the probability of codeword c_i being the last in an m-bit block is proportional to $p_i l_i$, and that for a given codeword, each bit position has the same chance to be the last in the block. Hence R, the average number of "redundant" bits per block, i.e., the average length of the prefix of the last codeword in the block if it was truncated, is given by

$$R = \frac{1}{L} \sum_i p_i l_i \left(\frac{1}{l_i} \sum_{j=0}^{l_i - 1} j \right) = \frac{1}{2} \left(\frac{\sum p_i l_i^2}{L} - 1 \right),$$

where $L = \sum p_i l_i$ is the original average codeword length.

The new average number of codewords per block is $N' = (m - R)/L$ and the new average codeword length is

$$L' = \frac{m}{N'} = \frac{mL}{m - R},$$

from which a bound for m can be derived, when a desired upper bound for a new average codeword length $L' > L$ is given: the new average codeword length will not exceed L' if

$$m \geqslant \frac{R \cdot L'}{L' - L}.$$

Once the block size is fixed, we proceed to calculate $\mathscr{SF}''$. Given that an error has occurred, the probability that this error is in the first codeword of an m-bit block is L/m. In this case, at most the entire block (N' codewords) is lost. If the error is in the second codeword of the block, at most $N' - 1$ codewords are lost, again with probability L/m, etc. Assuming that N' is an integer, we get

$$\mathscr{SF}'' = \frac{L}{m}(N' + (N' - 1) + \cdots + 1) = \frac{L(N' + 1)N'}{2m}.$$

The resulting formula $\mathscr{SF}'' = L(N' + 1)N'/2m$ is approximately true also for non-integral N'.

It is not always possible to achieve a predetermined $\mathscr{SF}''$ because m cannot be smaller than the maximal codeword length. For some distributions, one can obtain the same $\mathscr{SF}''$ as for some constant code $\mathscr{L}_n(\pi)$, but with larger average codeword length. For other distributions a block size can be found which gives both better $\mathscr{SF}''$ and better compression than $\mathscr{L}_n(\pi)$; in these cases the advantage of the latter reduces to their simplicity, their faster decoding and their robustness against insertion and deletion errors (see examples in Section 5). Nevertheless it should be noted that while $\mathscr{SF}''$ (and even $\mathscr{SF}$) for the $\mathscr{L}_n(\pi)$ codes is bounded, $\mathscr{SF}''$ for the "robustified" Huffman codes depends on the ratio of the maximum to the average codeword length, which in turn is a function of the number of elements of the set and their distribution. This ratio is minimized for the uniform distribution, but this is the worst case from the compression point of view.

An alternative way to protect Huffman codes against noise is proposed in [15, Section 4.14]: break the Huffman encoded message into blocks and use Hamming error-correcting codes to protect each block. If m is the size of the Huffman code blocks, the output blocks are of size $m + \lceil \log_2 m \rceil$. This can therefore be an attractive alternative, since for large enough m, compression is only slightly deteriorated, but $\mathscr{S}\mathscr{F}'' = 0$. On the other hand, the coding algorithms are much more complicated and time consuming.

3. Universality and completeness

The previous section has dealt with criterion (i) mentioned in the introduction. We now turn to the other two criteria. As was pointed out earlier, a simple way to encode an alphabet of n elements is to use the first n codewords of a fixed infinite code. In [6], Elias has shown that it is possible to construct infinite codeword sets which he calls universal: an infinite set of codewords of lengths l_i, with $l_1 \leqslant l_2 \leqslant \cdots$, is *universal* if for any finite probability distribution $P = (p_1, \ldots, p_n)$, with $p_1 \geqslant p_2 \geqslant \cdots$, the following inequality holds: $\sum_{i=1}^{n} p_i l_i / \max(1, E(P)) \leqslant K$, where $E(P) = -\sum_{i=1}^{n} p_i \log_2 p_i$ is the entropy of the distribution P, and K is a constant independent of P. Thus given any arbitrary probability distribution of an alphabet, a universal code can be used to encode it such that the resulting average codeword length is at most a constant times the optimal possible for that distribution.

The universality of the logarithmic ramp code $\mathscr{R}$ has been shown in [6]. The unary code $\mathscr{U}$ is not universal. The codes $\mathscr{L}(\pi)$ are universal if and only if π has at least 3 bits or $\pi = 11$ or $\pi = 00$ (see [22]).

Although we consider also the codes yielding suboptimal compression, we shall restrict ourselves to complete infinite codes. As defined in [6], a code C is *complete* if adding any binary string c, $c \notin C$, gives a set $C \cup \{c\}$ which is not UD. Other authors call such a code *succint* [25]. Note that an infinite code which is not complete can be extended by adjoining more codewords, thus forming a sequence with better compression capabilities.

Every UD code C with codeword lengths l_i satisfies the McMillan [24] inequality: $\sum_i 2^{-l_i} \leqslant 1$. Thus a sufficient condition for the completeness of C is $\sum_i 2^{-l_i} = 1$. In [22], recurrence relations are developed, giving for every fixed π the number $b_r(\pi)$ of elements of length r in $\mathscr{L}(\pi)$, for $r \geqslant k$. These relations can be used to show that for all such codes, $\sum_{r=k}^{\infty} b_r(\pi) 2^{-r} = 1$, which implies the completeness of the sets. An algebraic proof for the completeness of $\mathscr{L}(\pi)$ can be found in [3, Ch. II, Section 5]. We give here a direct, "string-theoretic" proof.

Theorem 1. *The codeword set $\mathscr{L}(\pi)$, generated by any fixed pattern π of $k \geqslant 1$ bits, is complete.*

Proof. Let $c = c_1 \ldots c_r$ be any binary string $\notin \mathscr{L}(\pi)$. In order to show that $\mathscr{L}(\pi)$ is complete, we construct a binary string which has more than one possible decomposition in the set $\mathscr{L}' = \mathscr{L}(\pi) \cup \{c\}$.

Let $\pi = \pi_1 \ldots \pi_k$ and define the string $E = c\pi = e_1 \ldots e_{r+k}$. Define a sequence of indices $t(i)$ for $i \geqslant 0$ by $t(0) = 0$ and for $i > 0$, $t(i)$ is such that $E(i) \overset{\text{def}}{=} e_{t(i-1)+1} \ldots e_{t(i)} \in \mathscr{L}(\pi)$. In other words, scanning the string E from left to right, we try to decompose it into elements of $\mathscr{L}(\pi)$, denoting by $t(i)$, for $i \geqslant 1$, the index of the last bit in E which belongs to the ith codeword detected in this way of scanning. Although π occurs as suffix in E, it is not always true that E can be decomposed in its entirety in this way. As example, take $\pi = 101$ and $E = 00101110101$, then $t(1) = 5$ and $t(2) = 9$ and we are left with a suffix 01 in E.

As can be seen in the example, the problem arises when there is an occurrence of π which has overlapping bits with the suffix π of E. Hence in this way, we parse E into $(E(1), \ldots, E(m), R)$ for some $m \geqslant 1$, where $E(i) \in \mathscr{L}(\pi)$ for $1 \leqslant i \leqslant m$ and R is a (possibly empty) proper suffix of π.

Case 1: R is empty. Then we have two decompositions of E in $\mathscr{L}'$: $E = (c, \pi) = (E(1), \ldots, E(m))$.

Case 2: R is not empty. Let $\bar{\pi}_1$ denote the binary complement of π_1 and let $b = b_1 \ldots b_{k-1}$ be the string defined by $b_i = \bar{\pi}_1$ for $1 \leqslant i < k$. Consider the string $B = Eb\pi$, one possible decomposition of which in $\mathscr{L}'$ is $(c, \pi, b\pi)$. A proper prefix of E can be parsed into $(E(1), \ldots, E(m))$ with $E(i) \in \mathscr{L}(\pi)$, so it remains to show that the suffix $S = Rb\pi$ can be decomposed into elements of $\mathscr{L}'$. If π occurs in S only as a suffix then $S \in \mathscr{L}(\pi)$. If π occurs twice in S, the two occurrences cannot be overlapping because of the choice of b; this yields a decomposition of S into two elements of $\mathscr{L}(\pi)$.

The proof is completed by showing that the pattern π cannot occur more than twice in S. Since R is a proper suffix of π, it has less than k bits, hence any occurrence of π which starts in R must extend into b. On the other hand, no occurrence of π can start in one of the bits of b. So if there are $h > 2$ appearances of π in S, one of the occurrences is as the suffix of S, and the $h - 1$ remaining occurrences of π must start at different positions in R, thus having suffixes of different lengths in b. However, this implies that all the bits of π are equal to $b_i = \bar{\pi}_1$, a contradiction. □

We saw already that as far as robustness is concerned, the pattern π for the code $\mathscr{L}(\pi)$ should be chosen with autocorrelation $10 \ldots 0$. Theorem 1 suggests that sets based on such patterns are preferable also in another sense. Extend Gilbert's block codes into a variable-length code in the following way (for technical reasons, we shall consider the codewords with fixed *suffix* rather than fixed *prefix*): for a fixed pattern π of length $k \geqslant 1$ bits, $\mathscr{G}(\pi)$ will denote the set of all the codewords of the form $y = x\pi$, where x is any binary string of length $\geqslant 0$, such that the pattern π occurs in $\pi x \pi$ only as prefix and suffix. Thus $\mathscr{G}(\pi)$ is the union for $N \geqslant k$ of all the block codes of length N as defined by Gilbert, except that we also permit the case $N = k$.

The code $\mathscr{G}(\pi)$ obtained in this way, which is comma free, is not the same as the code $\mathscr{L}(\pi)$, which is only UD; an example showing the difference, is the string 01000101 which is in $\mathscr{L}(0101)$ but not in $\mathscr{G}(0101)$. As the condition on the elements of $\mathscr{L}(\pi)$ is less restrictive than the condition on the elements of $\mathscr{G}(\pi)$, it follows that for any π, $\mathscr{G}(\pi) \subseteq \mathscr{L}(\pi)$.

Theorem 2. *The following assertions are equivalent:*

1. *The autocorrelation of π is of the form* 10 … 0.
2. $\mathscr{G}(\pi) = \mathscr{L}(\pi)$.
3. *The code $\mathscr{G}(\pi)$ is complete.*

Proof. ($1 \Rightarrow 2$) We know already that $\mathscr{G}(\pi) \subseteq \mathscr{L}(\pi)$ holds for every π; for the opposite inclusion, let $y = x\pi$ be a codeword in $\mathscr{L}(\pi)$, so that π appears in y only as suffix. If no proper prefix of π is also a suffix of π, then π occurs in $\pi y = \pi x \pi$ only as prefix and suffix, so that $y \in \mathscr{G}(\pi)$. Hence $\mathscr{G}(\pi) = \mathscr{L}(\pi)$.

($2 \Rightarrow 3$) This is Theorem 1. (For any π with autocorrelation 10 … 0, the proof of Theorem 1 is even much simpler, since Case 2 cannot occur.)

($3 \Rightarrow 1$) We show that if the autocorrelation of π is not 10 … 0, then $\mathscr{G}(\pi) \subset \mathscr{L}(\pi)$ holds with strict inclusion, so that $\mathscr{G}(\pi)$ cannot possibly be complete since $\mathscr{L}(\pi)$ is. Thus we look for a codeword B which is generated by π, but does not belong to $\mathscr{G}(\pi)$. Let $\pi = \pi_1 \ldots \pi_k$ and suppose that $\pi_1 \ldots \pi_h = \pi_{k-h+1} \ldots \pi_k$ for some $h < k$. Let $\bar{\pi}_i$ denote the binary complement of π_i and define the strings $b = b_1 \ldots b_{k-1}$ and ($d = d_1 \ldots d_{k-1}$ by $b_i = \bar{\pi}_1$ and $d_i = \bar{\pi}_k$ for $1 \leqslant i < k$. Consider the string $B = \pi_{h+1} \ldots \pi_k d b \pi$ which is not in $\mathscr{G}(\pi)$.

It remains to show that B is generated by π, or in other words that π occurs in B only as suffix. Because of the choice of b, π can occur in $b\pi$ only as suffix, and because of the choice of d, π cannot occur at all in $\pi_{h+1} \ldots \pi_k d$. As to the string db, assume first $\pi_1 = \pi_k$. Then db is a string of identical bits in which π can neither start nor end. Hence suppose $\pi_1 \neq \pi_k$. Then if π appears in db, it must be of the form $\pi = d_j \ldots d_{k-1} b_1 \ldots b_j$ for some $1 \leqslant j \leqslant k-1$, but in all these cases, the autocorrelation of π is 10 … 0, contradicting our assumption. □

4. Fibonacci codes

As was pointed out earlier, the first n elements of the code $\mathscr{L}(\pi)$, for certain patterns π, can be an attractive alternative to Huffman codes when optimal compression is not critical. The encoding process is simpler, since the code need not be generated for every probability distribution. However, except for the fact that a message encoded by $\mathscr{L}(\pi)$ is easily parsed by locating the separators π, the actual decoding algorithms are very similar for Huffman codes and $\mathscr{L}(\pi)$. For both, there is generally no simple relation between a codeword and its index, such as, e.g., for fixed-length codes or for the unary code $\mathscr{U}$. Therefore, one needs a “translation table”, which consists of two columns: one column containing the codewords, and the other containing the corresponding cleartext elements. For decoding, after having detected a codeword c, the algorithm searches for c in the column of codewords and retrieves then the corresponding element from the cleartext column. The existence of an easily computable one-to-one mapping between the code and the integers would make the column of codewords (and the search in it) superfluous. This means that the space requirements of the Huffman codes could be cut by $\frac{1}{2}$. It should however be noted that we refer here only to the straightforward approach to the decoding of Huffman codes. In certain cases, more sophisticated data structures may be used, which yield more efficient algorithms, as in [17, 5].

In this section, we study the code $\mathscr{L}(\pi)$ for the special case $\pi = 11$ and show that such a mapping exists, because the code is related to the binary Fibonacci numeration system. This relation has not been noted in [22], but has already been investigated in [1].

4.1. The Fibonacci code C^1

One can use a binary encoding of the integer i as encoding for the element A_i; if we are to use a fixed-length code, the length of the codewords will be $\lfloor \log_2 n \rfloor + 1$ for the standard binary numeration system. As we want a uniquely decipherable code, it is not possible to pass to a variable-length code by just omitting the leading zeros in every codeword, because of the resulting ambiguities. We propose to exploit a property of the binary Fibonacci numeration system: let F_j be the jth Fibonacci number,

$$F_0 = 0, \qquad F_1 = 1, \qquad F_j = F_{j-1} + F_{j-2} \quad \text{for } j > 1.$$

Then any integer i can be represented by the binary string $I = I_1 I_2 \ldots I_r$, with $I_j = 0$ or 1, where $i = \sum_{j=1}^{r} I_j F_{j+1}$. Note that the indexing in the string I increases from left to right, contrary to the usual notation; the reason for this will become clear in the sequel. One can uniquely express any integer in this form so that

$$I_j = 1 \;\Rightarrow\; I_{j-1} = 0 \quad \text{for } j = 2, \ldots, r,$$

in other words, there are no adjacent 1's in I. Although the number of bits needed to represent integers between 1 and n by fixed-length codes increases to $r = \lfloor \log_\phi(\sqrt{5}n) + 1 \rfloor$, where $\phi = (1 + \sqrt{5})/2$ is the golden ratio, we are now able to use a variable-length representation, replacing the trailing zeros in I by an additional 1 which will act as a "comma", separating consecutive codewords. We denote this infinite sequence of codewords by $C^1 = \{C_1^1, C_2^1, \ldots\} = \{11, 011, 0011, 1011, 00011, 10011, 01011, 000011, \ldots\}$, and the length of C_i^1 by l_i^1. The sequence C^1 is one of the possible orderings of $\mathscr{L}(11)$.

The properties of (generalized) Fibonacci numeration systems were used by Kautz [21] for synchronization control; some *fixed*-length codes were devised which satisfy the condition that every codeword contains no string of m or more consecutive 1's, for some fixed $m \geqslant 2$. The code C^1 extends this idea to *variable*-length codes, choosing $m = 2$ so that only one additional bit per codeword is needed to allow unique decipherability.

Remark. For the sake of completeness, we give direct proofs for the following propositions, some of which can be derived as special cases of the corresponding general proofs in [1].

Proposition 1. *There are F_r codewords of length $r + 1$ in C^1, $r \geqslant 1$.*

Proof. In the proposed representation, an integer j satisfying $F_{r+1} \leqslant j < F_{r+2}$ needs r bits for its encoding, $r \geqslant 1$, thus the claim follows if we add the "separating" 1 and note that $F_{r+2} - F_{r+1} = F_r$. □

Proposition 2. *The code C^1 is uniquely decipherable, universal and complete.*

Proof. After adding the "comma"-bit, every codeword terminates in two consecutive 1's, which cannot appear anywhere else in a codeword. Thus C^1 is a prefix code.

From Proposition 1 we get that the number of codewords of length up to and including r is $\sum_{i=1}^{r-1} F_i$, which by induction can be shown to equal $F_{r+1} - 1$. Thus if the length $l_i = l_i^1$ of C_i^1 is $r + 1$, the index i is at least $F_{r+1} = F_{l_i}$ so that $i \geqslant F_{l_i} > (1/\sqrt{5})\phi^{l_i} - 1$. Therefore, $l_i < \log_\phi(\sqrt{5}(i + 1)) < 3 + 2\log_2 i$. But since the p_i are arranged as a non-increasing sequence and sum to unity, we have $ip_i \leqslant \sum_{j=1}^{i} p_j \leqslant 1$, thus $p_i \leqslant 1/i$, so that $\log_2 p_i \leqslant -\log_2 i$. Hence

$$\sum p_i l_i < 3 + 2\sum p_i \log_2 i \leqslant 3 - 2\sum p_i \log_2 p_i = 3 + 2H(P).$$

Thus $K = 5$ can be chosen as constant in the definition of universality.

As to completeness, let us denote $\sum_{j=1}^{\infty} 2^{-l_i^1}$ by S. Using the Fibonacci recurrence relation, we get

$$\begin{aligned} S &= \sum_{i=1}^{\infty} 2^{-(i+1)} F_i = \sum_{i=1}^{\infty} 2^{-(i+1)}(F_{i+1} - F_{i-1}) \\ &= 2\sum_{i=2}^{\infty} 2^{-(i+1)} F_i - \tfrac{1}{2}\sum_{i=0}^{\infty} 2^{-(i+1)} F_i \\ &= 2(S - \tfrac{1}{4}) - \tfrac{1}{2}S = \tfrac{3}{2}S - \tfrac{1}{2}, \end{aligned}$$

thus $S = 1$, in other words, C^1 is complete. □

Note that if the conventional notation $I = I_r I_{r-1} \ldots I_1$ is used to represent an integer in the Fibonacci numeration system, and then the leading zeros are replaced by a 1 in the leftmost position, the resulting code is a suffix code, but not prefix. Hence the decoding procedure would be somewhat complicated as for each string of ones, we must know the parity of its length before we can interpret the codeword preceding the string.

To evaluate $\mathscr{S}\mathscr{F}(C^1)$, we first remark that $\pi = 11$ does not have autocorrelation 10. Nevertheless, $\mathscr{S}\mathscr{F}$ is bounded. The last three bits of every codeword (except C_1^1) are "011". If the error occurs elsewhere, only one codeword is lost (possibly one codeword will be interpreted as two), hence using the notations of Section 2.1, $M(C_i^1, j) = 1$ for $i > 1$, $1 \leqslant j \leqslant l_i - 3$. A problem may arise in case of an error in one of the three rightmost bits, if the codeword, the error occurs in, is followed by $j > 0$ consecutive C_1^1's. Suppose this string of j C_1^1's is followed by C_h^1 for $h > 1$, then the parsing of the encoded string up to and including C_h^1 could change. For example, 0011-11-11-011 would become 0011-11-11-1011 in case of insertion of 1 after one of the three rightmost bits; or it would become 00011-11-1011 by a substitution error in the penultimate bit; or it would become 011-11-11-1011 by a substitution error in the third bit from the right. However, our choice $\pi = 11$ differs from the example for $\pi = 11100111$ of Section 2.1, in that at least $j - 1$ of the codewords obtained by the incorrect parsing are C_1^1, so in the worst case, only the first codeword, at most one of the C_1^1, and C_h^1 are lost.

More precisely, an error in the rightmost bit of a codeword causes at most two codewords to be lost, hence $M(C_i^1, l_i) = 2$ for $i \geqslant 1$. An error in the penultimate bit may cause up to three false interpretations, i.e., $M(C_i^1, l_i - 1) = 3$ for $i \geqslant 1$. In case of an error in the third bit from the right, it may be possible to "decode" $j + 1 C_1^1$'s instead of j, as for example in 0011-11-1011 which becomes 011-11-11-011 by a substitution error, but only the first and last codewords are lost, i.e., $M(C_i^1, l_i - 2) = 2$ for $i > 1$.

Denote by $L_1 = \sum p_i l_i^1$ the average length of a codeword. Then we get from (1)

$$\mathcal{SF}(C^1) = \frac{1}{L_1}\left(p_1(2+3) + \sum_{i=2}^{n} p_i(2+3+2+(l_i^1 - 3))\right)$$

$$= \frac{1}{L_1}(5p_1 + 4(1 - p_1) + (L_1 - 2p_1)) = 1 + \frac{4 - p_1}{L_1}.$$

Numerical examples of $\mathcal{SF}(C^1)$ for various distributions are given in Section 5.

The decoding process of a message encoded by C^1 consists of two phases. First, the input string is parsed into codewords just by locating the separator "11". The index of each codeword is then evaluated and a table is accessed to translate the index to the corresponding cleartext element. The dominant part of the processing time is taken by this table access, which is much slower than the scanning of the first phase. On the other hand, for Huffman codes, even the first phase involves table or tree accesses for every bit, until a codeword is detected. Although for the same input S, the string $C^1(S)$ encoded by C^1 will be longer than the string $H(S)$ encoded by Huffman's algorithm, the number of *codewords* in $C^1(S)$, which is the number of times we have to access a table for the decoding of $C^1(S)$, will be much smaller than the number of *bits* in $H(S)$, which is the number of times we have to access a table or tree for the decoding of $H(S)$. We thus expect a faster decoding for C^1 than for Huffman codes. The relative savings will increase with the average codeword length for C^1 and with n, the size of the set of cleartext elements.

In the following algorithm, the encoded message is given in a bit vector M, the elements of which are denoted by M_i, $i = 1, 2, \ldots$. The algorithms for both encoding and decoding use a translation table in which the cleartext element A_j is stored at entry j, $1 \leqslant j \leqslant n$. Given a codeword c, we compute its index in C^1 using the Fibonacci numeration system, and can then directly access the translation table at the appropriate entry. The computation can be speeded up by the use of a table of Fibonacci numbers F_k.

```
**Decoding procedure for** $C^1$
$i \leftarrow 0$   ⟦pointer in $M$⟧
**while** $i <$ length($M$) **do**
  $k \leftarrow 2$
  $index \leftarrow 0$
  $i \leftarrow i + 1$   ⟦$i$ points to the leftmost bit of a codeword⟧
  **repeat**   ⟦evaluate index of the codeword⟧
    $index \leftarrow index + (M_i \times F_k)$
    $k \leftarrow k + 1$
    $i \leftarrow i + 1$
  **until** $M_{i-1}M_i = 11$   ⟦look for pattern "11'⟧
  access translation table at *index*
**end**
```

4.2. Higher-order Fibonacci codes

The idea of the previous subsection is easily generalized to higher-order Fibonacci codes. Fibonacci numbers of order $m \geqslant 2$ are defined by the recurrence

$$F_j^{(m)} = F_{j-1}^{(m)} + F_{j-2}^{(m)} + \cdots + F_{j-m}^{(m)} \quad \text{for } j > 1,$$

where $F_1^{(m)} = 1$ and $F_j^{(m)} = 0$ for $j \leqslant 0$. In particular, $F_j \equiv F_j^{(2)}$ are the standard Fibonacci numbers. As before, any integer i can be represented as a binary string $I = I_1 \ldots I_r$ such that $i = \sum_{j=1}^{r} I_j F_{j+1}^{(m)}$ and there is no run of m or more consecutive 1's in I. This fact is used in [1] to devise variable-length codes in which an m-bit run of 1's is used as a separator. Proofs that these "m-ary Fibonacci codes" are UD, universal and complete are also given in [1].

Using higher-order Fibonacci codes might at a first glance seem inefficient, particularly for the first codewords (corresponding to higher probabilities), because more bits are used as delimiters, so less bits carry actual information. On the other hand, with increasing m, the number of possible codewords of any fixed length increases. Hence for a large enough language to be encoded and for certain (near to uniform) distributions, it is possible to obtain an average codeword length, $L(m) = \sum p_i l_i(m)$, which is smaller for $m > 2$ than for $m = 2$. Here $l_i(m)$ denotes the length of the ith codeword of the m-ary Fibonacci code. The first line of Table 1 gives for a few $m > 2$ the minimal size $N(m)$ of the language for which the m-ary Fibonacci code yields an average codeword length not larger than that of code C^1, supposing uniform distributions, that is $N(m) = \min\{t \mid \sum_{i=1}^{t} l_i(2) \geqslant \sum_{i=1}^{t} l_i(m)\}$. For other distributions, the transition points, if they exist at all, would be higher.

Table 1
Comparison of C^1 with m-ary Fibonacci codes

m	3	4	5	6	7
$N(m)$	158	687	2972	12821	57626
$\mathcal{SF}(C^1)$ for $n = N(m)$	1.412	1.315	1.254	1.213	1.183
$\mathcal{SF}$(m-ary) for $n = N$(m)	1.618	1.630	1.636	1.639	1.639

By similar arguments as for C^1, one gets for the m-ary Fibonacci codes,

$$\begin{aligned}\mathcal{SF} &= \frac{1}{L(m)}\left(p_1(2+3(m-1)) + \sum_{i=2}^{n} p_i(2+3(m-1)+2+l_i(m)-(m+1))\right)\\ &= \frac{1}{L(m)}\left((2+3(m-1))p_1 + 2m(1-p_1) + (L(m)-mp_1)\right)\\ &= 1 + \frac{2m-p_1}{L(m)}.\end{aligned}$$

The second line of Table 1 depicts the $\mathcal{SF}$ of the standard code C^1 for a uniform distribution on a language of size $N(m)$, whereas the last line gives the $\mathcal{SF}$ of the m-ary Fibonacci code for the same distributions.

The table shows that compression is improved for higher-order codes only for fairly large sizes of the language.

4.3. *Variants based on Fibonacci codes of order 2*

In C^1, a 1-bit playing the role of a comma was added at the end of every codeword. This additional bit can be avoided if every codeword has a 1 not only in its rightmost position, but also in its leftmost. A new code C^2 is generated from C^1 by:

1. deleting the rightmost (1-)bit of every codeword;
2. dropping the codewords in C^1 which start with 0.

Another way to obtain the same set from C^1 is by:

1. deleting the rightmost (1-)bit of every codeword;
2. prefixing every codeword by 10;
3. adding 1 as the first codeword.

The equivalence of these two definitions is established by noting that the function $f(a_1 \dots a_r) = 10a_1 \dots a_{r-1}$ defines for both definitions of C^2 a one-to-one mapping from C^1 onto $C^2 - \{1\}$. Hence C^2 is the set of codewords $\{1, 101, 1001, 10001, 10101, 100001, 101001, \dots\}$. Their respective lengths are denoted l_i^2. We therefore have as immediate consequence of Proposition 1 the following proposition.

Proposition 3. *In C^2, there is one codeword of length* 1 *and there are F_{n-2} codewords of length n for $n \geqslant 3$.*

If a substitution error occurs in C_1^2, which consists of a single bit, the preceding and following codewords join up, in which case three codewords are lost; deletion and insertion affect only one codeword, so $M(C_1^2, 1) = 3$. For other codewords, a substitution or deletion error in the first or last bit causes the loss of two codewords, thus $M(C_i^2, 1) = M(C_i^2, l_i^2) = 2$ for $i > 1$; in the other cases, a single codeword is lost, $M(C_i^2, j) = 1$ for $i > 1$ and $1 < j < l_i^2$. Denoting now the average codeword length by L_2, we get

$$\mathcal{SF}(C^2) = \frac{1}{L_2}\left(3p_1 + \sum_{i=2}^{n} p_i(2 + (l_i^2 - 2) + 2)\right)$$

$$= \frac{1}{L_2}(3p_1 + 2(1 - p_1) + (L_2 - p_1)) = 1 + \frac{2}{L_2}.$$

Thus for distributions for which $L_1/L_2 < 2 - p_1/2$, and in particular when $L_1 = L_2$, C^2 is more robust. Note that C^2 is not a prefix code; nevertheless decoding is simple since the end of any codeword is easily detected.

Proposition 4. *The code C^2 is uniquely decipherable, universal and complete.*

Proof. Let M be an ambiguous encoding of a message, $M = c_1c_2 \ldots = c'_1c'_2 \ldots$, where $c_i, c'_j \in C^2$, and $M_1, M_2, \ldots$ are the bits the encoded message consists of. Let j be the smallest index for which $c_j \neq c'_j$. Then necessarily $|c_j| \neq |c'_j|$, suppose $|c_j| < |c'_j|$. Let a be the index of the rightmost bit in c_j. Then $M_{a+1} = 1$ since this is the first bit of c_{j+1}. But M_a is the last bit of c_j, hence $M_a = 1$ so that c'_j contains adjacent 1's, a contradiction. Hence C^2 is UD. The construction of C^2 implies that the lengths of the elements of C^1 and C^2 are related by $l_i^2 = l_{i-1}^1 + 1$ for $i > 1$. Therefore,

$$\sum_{i=1}^{n} p_i l_i^2 = p_1 + \sum_{i=1}^{n-1} p_{i+1}(l_i^1 + 1) \leqslant 1 + \sum_{i=1}^{n} p_i l_i^1$$

so that the universality of C^2 follows from that of C^1. As to completeness,

$$\sum_{i=1}^{\infty} 2^{-|c_i^2|} = \frac{1}{2} + \sum_{i=1}^{\infty} 2^{-(i+2)}F_i = \frac{1}{2} + \frac{1}{2}\sum_{i=1}^{\infty} 2^{-(i+1)}F_i = 1,$$

the last sum being the quantity S of Proposition 2. □

The decoding algorithm again searches for the occurrence of the pattern "11", which is formed by juxtaposing any two codewords. A special treatment of the last codeword is avoided by suffixing an additional "1" at the end of the input string. The function which maps a codeword (except the first) into its index simply ignores the first two bits ("10") and proceeds then as for C^1.

```
Decoding procedure for $C^2$
$N \leftarrow$ length$(M)$
$M_{N+1} \leftarrow 1$   ⟦suffixing 1 at the end of the input string⟧
$i \leftarrow 1$
while $i \leqslant N$ do   ⟦$i$ points to the leftmost bit of a codeword⟧
  if $M_i M_{i+1} = 11$ then   ⟦codeword "1"⟧
    access translation table at first entry
    $i \leftarrow i + 1$
  else
    $index \leftarrow 1$
    $i \leftarrow i + 2$   ⟦skip 10⟧
    $k \leftarrow 2$
    repeat   ⟦evaluate index of the codeword⟧
      $index \leftarrow index + (M_i \times F_k)$
      $k \leftarrow k + 1$
      $i \leftarrow i + 1$
    until $M_{i-1} M_i = 11$   ⟦look for pattern "11"⟧
    access translation table at $index$
end
```

Generalizations of the code C^2 to higher-order Fibonacci codes are given in [1].

Another attempt to avoid the comma-bit in C^1 is to construct a new sequence C^3 of codewords, which is obtained from C^1 by:

1. deleting the rightmost (1-)bit of every codeword;

2. duplicating the set of codewords of length r, for every $r \geqslant 1$; now we have for each r two identical blocks of codewords;

3. prefixing in the first block every codeword by "10" and in the second by "11".

This yields the set of codewords $C^3 = \{101, 111, 1001, 1101, 10001, 10101, 11001, 11101, 100001, 101001, 100101, 110001, \ldots\}$, their lengths are denoted l_i^3. Note that every codeword of C^3 has a leftmost 1-bit, no codeword has more than 3 consecutive 1-bits and these appear as prefix, and every codeword, except C_2^3, terminates in "01". From the construction of C^3 and Proposition 1 we get the following proposition.

Proposition 5. *In C^3, there are $2F_{r-2}$ codewords of length r for $r \geqslant 3$.*

A substitution error in the first bit of C_2^3 affects also the preceding and the following codeword, so there are three codewords lost. Any other error in this bit, as well as any error in the other bits of C_2^3, causes the loss of up to two codewords. In the other codewords (including C_1^3), an error in the first, last and penultimate bit causes up to two incorrect interpretations, elsewhere one. Setting $L_3 = \sum p_i l_i^3$, we get

$$\mathcal{S}\mathcal{F}(C^3) = \frac{1}{L_3}\left(p_2(3+2+2) + \sum_{i=1, i\neq 2}^{n} p_i(2+(l_i^3-3)+2+2)\right)$$

$$= \frac{1}{L_3}(7p_2 + (3-3p_2) + (L_3 - 3p_2))$$

$$= 1 + \frac{3+p_2}{L_3}.$$

Thus for distributions for which $L_3 = L_1$, C^3 is more robust than C^1, but for distributions for which $L_3 = L_2$, C^2 is more robust than C^3. The set C^3 too is not prefix, but we do have the following result.

Proposition 6. *The code C^3 is uniquely decipherable, universal and complete.*

Proof. We use the same notations as in Proposition 4. The codeword c_j cannot be $C_2^3 = 111$ since if it is, then there are four consecutive 1's in M (the three of c_j and the first of c_{j+1}), thus c'_j must also be C_2^3, but j was chosen such that $c_j \neq c'_j$. Any other codeword has a 0 in the penultimate position. Thus c'_j contains the pattern "011", which is impossible, hence C^3 is UD. Universality follows from the fact that $l_i^3 \leqslant l_i^1$ for $i > 1$. By Proposition 5, completeness follows from

$$\sum_{i=1}^{\infty} 2^{-|C_i^3|} = 2\sum_{i=1}^{\infty} 2^{-(i+2)}F_i = \sum_{i=1}^{\infty} 2^{-(i+1)}F_i = 1,$$

as was shown in the proof of Proposition 2. □

For decoding, after having checked that the codeword is not 111, we search for the pattern "011". As before, we add a "1" at the end of the input to allow identical processing of all the codewords. The index of a codeword of length r of the form $y_1 y_2 \ldots y_r$ (recall that $y_r = 1$) is computed by adding together the following three quantities: (a) The number of codewords of length $< r$, which is $\sum_{i=3}^{r-1} 2F_{i-2} = 2F_{r-1} - 2$; (b) $y_2 F_{r-2}$, since depending on the value of y_2 a codeword belongs to one of the two blocks, each of size F_{r-2}, which are defined in step 2 of the construction of C^3; (c) the relative index within the block. This relative index is obtained by considering the $r-2$ rightmost bits of the codeword as the representation of an integer in the Fibonacci numeration system, and subtracting $F_{r-1} - 1$ since the $r-2$ rightmost bits represent integers in the range $[F_{r-1}, F_r - 1]$. Summarizing,

$$\textit{index} = 2F_{r-1} - 2 + y_2 F_{r-2} + \sum_{i=3}^{r} y_i F_{i-1} - F_{r-1} + 1$$

$$= \sum_{i=3}^{r+1} y_i F_{i-1} + (y_2 - 1)F_{r-2} - 1,$$

where $y_{r+1} = 1$ is the first bit of the following codeword.

```
Decoding procedure for $C^3$
$N \leftarrow$ length($M$)
$M_{N+1} \leftarrow 1$   ⟦suffixing "1" at the end of the input string⟧
$i \leftarrow 2$
while $i < N$ do   ⟦$i$ points to the 2nd bit from the left of a codeword⟧
  if $M_{i-1}M_iM_{i+1} = 111$ then   ⟦codeword 111⟧
    access translation table at second entry
    $i \leftarrow i + 3$
  else
    $index \leftarrow -1$
    $y_2 \leftarrow M_i$   ⟦second bit⟧
    $k \leftarrow 1$
    repeat   ⟦evaluate index of the codeword⟧
      $i \leftarrow i + 1$
      $k \leftarrow k + 1$
      $index \leftarrow index + (M_i \times F_k)$
    until $M_{i-2}M_{i-1}M_i = 011$   ⟦look for pattern "011"⟧
    access translation table at $index + (y_2 - 1)F_{k-2}$
    $i \leftarrow i + 1$
  end
```

5. Examples

Three "real-life" examples were chosen, each showing the optimality of another variant for the given distribution. The first example is the distribution of the 26 characters in an English text of 100000 words chosen from many different sources, as given by Heaps [16]. In Table 2, the letters are listed in decreasing probability of occurrence, together with their Huffman code, C^1, C^2 and C^3 codes. For the Huffman code, the codewords for the letters L and K are synchronizing. This is the example in [8] of the Huffman code for English which maximizes the sum of the probabilities of the synchronizing codewords; finding the *best* possible Huffman code in this sense is still an open problem.

The second example is the distribution of 30 Hebrew letters (including two kinds of apostrophes and blank) as computed from the data base of the Responsa Retrieval Project [9] of about 40 million Hebrew and Aramaic words. Using the method presented in [8], we constructed a Huffman code for this alphabet with one synchronizing codeword, which appeared with probability 0.0035.

The third example is of a different kind. A large sparse bit-vector may be compressed in the following way (see, for example, [19]): the vector is partitioned into

Table 2
Distribution of letters in English text

Letter	Probability	Huffman	C^1	C^2	C^3
E	0.1265	011	11	1	101
T	0.0978	111	011	101	111
A	0.0789	0001	0011	1001	1001
O	0.0776	0011	1011	10001	1101
I	0.0707	0100	00011	10101	10001
N	0.0706	0101	10011	100001	10101
S	0.0631	1010	01011	101001	11001
R	0.0595	1011	000011	100101	11101
H	0.0574	1100	100011	1000001	100001
L	0.0394	11011	010011	1010001	101001
D	0.0389	11010	001011	1001001	100101
U	0.0280	10011	101011	1000101	110001
C	0.0268	10010	0000011	1010101	111001
F	0.0256	00101	1000011	10000001	110101
M	0.0244	00100	0100011	10100001	1000001
W	0.0214	00001	0010011	10010001	1010001
Y	0.0202	000000	1010011	10001001	1001001
G	0.0187	000001	0001011	10101001	1000101
P	0.0186	100001	1001011	10000101	1010101
B	0.0156	100010	0101011	10100101	1100001
V	0.0102	100011	00000011	10010101	1110001
K	0.0060	1000001	10000011	100000001	1101001
X	0.0016	10000001	01000011	101000001	1100101
J	0.0010	100000001	00100011	100100001	1110101
Q	0.0009	1000000001	10100011	100010001	10000001
Z	0.0006	1000000000	00010011	101010001	10100001
Weighted average		4.185	4.895	5.298	4.891

k-bit blocks, then the 2^k possible block patterns are assigned Huffman (or other) codes according to their probability or occurrence. The statistics were collected from 15 378 bit-vectors of 42 272 bits each, which were constructed at the Responsa Project: each vector serves as an "occurrence map" for a different word, the bit position referring to the number of the document, where the value at position i is 1 if and only if the given word appears in the ith document. We chose $k = 8$, thus the alphabet consisted of 256 "characters". As the vectors are extremely sparse – the proportion of 1-bits is only 1.7% – the probability of a block consisting only of zeros is high (0.925), hence there is much waste in using a code such as C^1 or C^3, for which the first codeword is longer than one bit. By [8, Theorem 5], the Huffman code corresponding to this distribution is synchronous, the only synchronizing codeword we found had probability 0.000048. (Actually, using the notion of generalized numeration systems, one can achieve much better compression of sparse bit-vectors than the Huffman compression approach of [19]! See [12].)

Table 3
Average values for 1000 coded characters

		English 26 letters	Hebrew 30 letters	Bit-vectors 256 letters
Huffman	length	4185	4285	1415
	$\mathcal{SF}'$	*14.84*	*166.5*	*18259*
C^1	length	4895	4824	2326
	$\mathcal{SF}$	1.849	1.874	2.436
C^2	length	5298	5127	1450
	$\mathcal{SF}$	1.551	1.653	2.876
C^3	length	4891	4884	3235
	$\mathcal{SF}$	1.633	1.632	1.929

Table 4
Average values using m-bit blocks

m	English		Hebrew		Bit-vectors	
	length	$\mathcal{SF}''$	length	$\mathcal{SF}''$	length	$\mathcal{SF}''$
9		—	5408	1.056		—
10	5041	1.238	5270	1.178		—
11	4949	1.362	5162	1.300		—
12	4875	1.486	5075	1.420	1559	3.945
13	4814	1.608	5004	1.540	1547	4.299
14	4763	1.731	4945	1.660	1537	4.653
15	4720	1.853	4895	1.779	1528	5.007
16	4682	1.974	4852	1.898	1520	5.361
17	4650	2.095	4814	2.017	1514	5.715
18	4621	2.217	4781	2.135	1508	6.069
19	4596	2.338	4752	2.253	1503	6.422
20	4574	2.458	4727	2.371	1498	6.776
50	4332	6.058	4451	5.888	1447	17.383
100	4258	12.036	4367	11.727	1431	35.056
∞	4185	*14.843*	4285	*166.5*	1415	*18259*

Table 3 summarizes the results. The lines headed “length” give the expected length in bits of a file of 1000 coded characters. The sensitivity factors were computed using the given probability distributions. For the Huffman codes, the table gives the sensitivity factor $\mathcal{SF}'$ and their values are italicized to differentiate them from the $\mathcal{SF}$-values. If fixed-length codes were used, 5000 bits would be necessary for the English or Hebrew alphabet and 8000 bits for the bit-vector.

Table 4 gives the new values for the length and $\mathcal{SF}''$ when m-bit blocks are used to improve the robustness of Huffman codes. These values were computed using the formulae of Section 2.3. The line for $m = \infty$ corresponds to the original Huffman algorithm, again with $\mathcal{SF}'$.

As can be seen, for the English alphabet with $m \in \{12, 13\}$, both the $\mathscr{SF}''$ and the average length are better than for C^3, which was the best of the C^i codes for this example. For the Hebrew alphabet the code C^1 always gives either better compression or better robustness and for $m = 16$ both values are better. The bit-vectors are an example of a case where a value of $\mathscr{SF}''$ as good as the $\mathscr{SF}$ for the C^i codes cannot be reached, since m must not be smaller than 12 which was the length of the longest codeword. Moreover, for small values of m, both $\mathscr{SF}''$ and length are worse than for C^2 and only for $m \geqslant 47$ the average length is shorter than for C^2, but with $\mathscr{SF}''$ as high as 15.34.

6. Concluding remarks

New sequences of variable-length codes were proposed, for applications where Huffman codes cannot be applied, e.g., when the probability distribution is not exactly known or changes in time, and for situations where the optimal compression of Huffman codes is not critical, and simplicity, faster processing and robustness against errors are preferred. If we restrict ourselves to a model allowing only substitution errors (as in [15] and in Section 2.3), then the simplest way to obtain the above properties is to use fixed-length codes, which however are independent of the probability distribution and may thus be very inefficient. The C^i codes proposed here, which can be encoded and decoded very efficiently, both in time and space, should then be regarded as a compromise between fixed-length and Huffman codes.

However, since our definition of an error allows also the *number* of transmitted bits to be changed, a fixed-length code F becomes even more vulnerable than some Huffman codes. An additional bit or a lost bit cause a shift of the encoded string, which will therefore be incorrectly interpreted, so that $\mathscr{SF}(F)$ will not be bounded when the number of encoded cleartext elements grows indefinitely. Although a single bit error may be self-correcting after a few codewords for certain codes, there are many others (e.g., when all the codewords have even length) for which this is not possible when the number of transmitted bits changes. On the other hand, the C^i codes are immune also to such errors, the number of false interpretations being still at most 3.

References

[1] A. Apostolico and A.S. Fraenkel, Robust transmission of unbounded strings using Fibonacci representations, IEEE Trans. Inform. Theory 33 (1987) 238–245.
[2] T. Bell, J.G. Cleary and I.H. Witten, Text Compression (Prentice-Hall, Englewood Cliffs, NJ, 1990).
[3] J. Berstel and D. Perrin, Theory of Codes (Academic Press, Orlando, FL, 1985).
[4] A. Bookstein and S.T. Klein, Is Huffman coding dead?, Computing 50 (1993) 279–296.
[5] Y. Choueka, S.T. Klein and Y. Perl, Efficient variants of Huffman codes in high level languages, in: Proceedings of the 8th ACM-SIGIR Conference, Montreal (1985) 122–130.
[6] P. Elias, Universal codeword sets and representation of the integers, IEEE Trans. Inform. Theory 12 (1975) 194–203.

[7] S. Even and M. Rodeh, Economical encoding of commas between strings, Comm. ACM 21 (1978) 315–317.

[8] T.J. Ferguson and J.H. Rabinowitz, Self-synchronizing Huffman codes, IEEE Trans. Inform. Theory 30 (1984) 687–693.

[9] A.S. Fraenkel, All about the responsa retrieval project you always wanted to know but were afraid to ask, expanded summary, Jurimetrics J. 16 (1976) 149–156.

[10] A.S. Fraenkel, Systems of numeration, Amer. Math. Monthly 92 (1985) 105–114.

[11] A.S. Fraenkel, The use and usefulness of numeration systems, Inform. and Comput. 81 (1989) 46–61.

[12] A.S. Fraenkel and S.T. Klein, Novel compression of sparse bit-strings – preliminary report, Combinatorial Algorithms on Words, NATO ASI Series F12 (Springer, Berlin, 1985) 169–183.

[13] E.N. Gilbert, Synchronization of binary messages, IRE Trans. Inform. Theory 6 (1960) 470–477.

[14] L.J. Guibas and A.M. Odlyzko, Maximal prefix-synchronized codes, SIAM J. Appl. Math. 35 (1978) 401–418.

[15] R.W. Hamming, Coding and Information Theory (Prentice-Hall, Englewood Cliffs, NJ, 2nd ed., 1986).

[16] H.S. Heaps, Information Retrieval, Computational and Theoretical Aspects (Academic Press, New York, 1978).

[17] D.S. Hirschberg, and D.A. Lelewer, Efficient decoding of prefix codes, Comm. ACM 33 (1990) 449–459.

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

[19] M. Jakobsson, Huffman coding in bit-vector compression, Inform. Process. Lett. 7 (1978) 304–307.

[20] B.H. Jiggs, Recent results in comma-free codes, Canad. J. Math. 15 (1963) 178–187.

[21] W.H. Kautz, Fibonacci codes for synchronization control, IEEE Trans. Inform Theory 11 (1965) 284–292.

[22] K.B. Lakshmanan, On universal codeword sets, IEEE Trans. Inform. Theory 27 (1981) 659–662.

[23] D.A. Lelewer and D.S. Hirschberg, Data compression, ACM Comput. Surveys 19 (1987) 261–296.

[24] B. McMillan, Two inequalities implied by unique decipherability, IRE Trans. Inform. Theory 2 (1956) 115–116.

[25] J.A. Storer, Data Compression: Methods and Theory (Computer Science Press, Rockville, MD, 1988).

[26] R.N. Williams, Adaptive Data Compression (Kluwer Academic Publishers, Dordrecht, 1990).

[27] E. Zeckendorf, Représentation des nombres naturels par une somme de nombres de Fibonacci ou de nombres de Lucas, Bull. Soc. Roy. Sci. Liège 41 (1972) 179–182.

[28] J. Ziv and A. Lempel, A universal algorithm for sequential data compression, IEEE Trans. Inform. Theory 23 (1977) 337–343.

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