Data Compression for a Source with Markov Characteristics

J. A. Llewellyn
Department of Computing, Engineering Building, University of Lancaster, Bailrigg, Lancaster LA1 4 YR

The entropy of an information source gives a lower limit to the data compression, measured as the average code/source ratio, which can be achieved without loss of information when transmitting or storing data from that source. The Huffman method for the design of a coding transformation guarantees optimum compression for a finite approximation to the source, i.e. a fixed number of source symbols (or symbol groups) and their respective probabilities. By taking sufficiently large groups, the code/source ratio can be decreased arbitrarily near to the limiting entropy.

The method to be presented in this paper takes a contrary approach and guarantees optimum compression for a preselected number of available code symbols (or symbol groups) and takes into account the statistical structure of the source in an optimum way. The use of finite state models provides a formal representation of the resulting encoding transformation, from which encoders and decoders can be designed.

The method is further extended to allow for exclusion of source strings with zero probability, but loses the absolute guarantee of optimum data compression for the given number of available code symbols.

Received May 1985

1. INTRODUCTION

In communicating data, messages are generated from a source alphabet, transformed into an appropriate code alphabet for transmission (often binary) and finally decoded into the source alphabet to deliver the original message at the receiver. The storage of data in a computer system can be considered in a similar way, in that a file of data, in some user alphabet, is transformed into the alphabet of the storage medium (usually binary), prior to writing to store. The data is subsequently decoded into its original form, in the source alphabet, when the file is read from store. The model depicted in Fig. 1 represents both applications.

![Diagram of communication model]

Figure 1. A communication model

One objective in designing the encoding transformation is to use the code alphabet efficiently, in order to economise on transmission time or storage space. It is well known, from Information Theory, that optimum use of a source is achieved when its symbols are generated uniformly without restriction. Occurrences of symbols from a source alphabet as the input to an encoder are frequently far from uniform, as for example in transmitting a random sequence of binary-coded decimal digits for which symbol probabilities are $P(0) = 5/8 \text{ and } P(1) = 3/8$. Similarly, in the use of natural languages, as in English text or formal languages, such as those used for writing computer programs, patterns of usage occur and restrictions are placed on the use of the alphabet by the grammar which defines the language. It is clearly desirable to design an encoding process which transforms sequences of symbols from the source so that disproportionate use is smoothed and a uniform sequence of code symbols is produced.

The capability of a source to communicate information is defined by the Entropy, a function of the probabilities and the statistical properties associated with the symbols of the source. In many cases the statistical nature of the source cannot be predicted precisely. However, estimates which approximate to the behaviour of the source can be made, based on the observation of a large sample of messages from that source. For example, as a first approximation, the source symbol probabilities can be estimated, ignoring restrictions and recurring patterns. That is, the source is regarded as zero memory and the number of occurrences of each symbol is counted. In simple English text, approximately 10% of symbols are 'E'. As a better approximation, recurring patterns and restrictions can be allowed for in estimating the probabilities by considering the source as a more complex Markov source. For example, in generating English text, if the state is that 'Q' has been received at the input to an encoder it is almost certain that the next symbol will be 'U'. Markov sources may be ergodic, i.e. independent of the time from the first observation, or they may be non-ergodic, in which case they have statistically transient behaviour. Markov sources will be introduced formally and examples of such sources will be given to illustrate and investigate coding methods. Finally, a method for the design of encoding transformations, which will take account of unequal use of source symbols and other restrictions placed on the source alphabet, will be formally developed, proved and illustrated. The result will be optimum, the most efficient possible for the preselected number of code symbols and for the given source specification.

2. DESIGN OF AN ENCODING TRANSFORMATION

Shannon's first theorem for noiseless coding, on the assumption that encoded messages are not subject to errors during transmission, states that the entropy of a source gives an upper limit for the efficiency of encoding
messages from the source. Moreover, by taking sufficiently large groups of source symbols, i.e., sufficiently high extensions or adjoints, it is possible to approach that limit as closely as desired. The Code to Source Ratio (CSR) is defined as the average number of code symbols required for each source symbol. The efficiency of the code transformation is \( Ef = \left(\frac{\text{Entropy}}{\text{CSR}}\right) \times 100 \), where the entropy units correspond to the number of symbols in the code alphabet. Unless otherwise stated, a binary code alphabet will be assumed, with entropy in bits per source symbol. From Shannon's theorem it is clear that \( Ef \leq 100\% \). Consider the design of encoding transformations into a binary alphabet for two illustrative sources. The simplest method is to allocate a fixed-length binary codeword to each symbol of the source or, alternatively, to each fixed-length group of source symbols, say symbol pairs.

### 2.1 Examples of information source coding

(a) A zero-memory binary source A, for which symbol probabilities are \( P(0,1) = (0.25, 0.75) \). Since the source is zero memory, symbols occur independently. Entropy \( H = 0.8113 \) bits per source symbol.

- **Code 1.** [0 1] → [0 1] \n  \( \text{CSR} = 1.0 \) \( Ef = 81\% \)

- **Code 2.** [00 01 10 11] → [00 01 10 11] \n  \( \text{CSR} = 1.0 \) \( Ef = 81\% \)

(b) A four-symbol source B with an alphabet \{a, b, c, d\}. Source sequences from the set \{aa ad bb bd ca cb dc dd bcc ccc\} are never permitted to occur. Otherwise if choice of next symbol is allowed, all symbols are equally probable. Entropy \( H = 0.56 \) bits per source symbol.

- **Code 1.** [a b c d] → [00 01 10 11] \n  \( \text{CSR} = 2.0 \) \( Ef = 28\% \)

- **Code 2.** [ab ac ba bc cc cd db] → [000 001 010 011 100 101 110].

(Due to source restrictions only seven symbol pairs are permitted.) \( \text{CSR} = 1.5 \) \( Ef = 37\% \) It is clear that no improvement is achieved by using the simple code transformation on higher extensions of source A. Limited improvements are achieved in the case of source B. To achieve further improvement, probabilities of symbol use need to be taken into account and symbol groups of variable length used. It is desirable that an encoding transformation be based on an instantaneous code, i.e., a code such that no codeword is a prefix of any other codeword in the set. Such codes can be defined by deterministic finite-state models, and recognition of a codeword is achieved without delay. Two basic approaches (or combinations of the two) may be used to design encoding transformations:

- Fixed length message → Variable length code; and
- Variable length message → Fixed length code.

### 2.2 Fixed-length messages

Huffman produced a method for the design of encoding transformations which produces compact variable-length codes, given a set of source messages and their probabilities. It can be found in most texts on coding (see Abramson). It has been used extensively to compact data sets, e.g., Wells and Yip. Wells compares two methods for transformation from fixed- to variable-length codewords, software using a table lookup technique and a hardware decision circuit. Yip uses Huffman's method to produce a finite-state model of the encoding transformation, using software to convert the output from the finite-state model into fixed-length blocks before transferring to store. In neither case has the problem of data sources with high-order Markov characteristics been discussed by the author.

The Huffman coding method considers symbols (or fixed-length groups of symbols from a source adjacent) together with their probabilities. A coding tree is constructed with the objective of producing an evenly distributed set of code symbols which are distributed with equal frequency in the corresponding variable-length codewords. The method guarantees to produce a compact code, i.e., a code with a minimum CSR for that particular source—symbol grouping. The CSR approaches the theoretical optimum as higher extensions are considered but the coding complexity increases exponentially. It does not take into account any interaction between groups of symbols, and the best encoding transformation to be expected is limited by the source approximation. Source entropy is overestimated if symbol interaction is not taken fully into account, so that encoding fixed-length groups from the adjoint approximation can limit the efficiency which can be achieved. This is particularly unhelpful when considering languages which are highly structured, as illustrated in the Source B example.

**Examples of Huffman coding of information sources.**

(a) **Source A**, second extension:

\[ P(00 01 10 11) = (1/6 3/16 3/16 9/16) \]

Using the Huffman method:

\[ [00 01 10 11] \rightarrow [000 001 001] \]

\( \text{CSR} = 0.844 \) \( Ef = 96\% \)

(b) **Source B**, second extension:

\[ P(ab ac ba bc cc cd db) = (0.08 0.08 0.16 0.16 0.047 0.233 0.24) \]

Using the Huffman method:

\[ [ab ac ba bc cc cd db] \rightarrow [0001 001 100 101 0000 0111] \]

\( \text{CSR} = 1.327 \) \( Ef = 42\% \)

Two major disadvantages of the Huffman method are the following:

(a) Computation time and space increase exponentially as higher-order extensions or adjoints are considered: for high-order Markov sources, high-order adjoints need to be used in order to take source structure into account.

(b) Data units used in transmission or storage are frequently of fixed size, so that variable-length codes produced by the method need to be blocked into appropriate size groups.

### 2.3 Variable length messages

Practical applications of file compression have been reported by a number of authors, in which variable-length text substrings are identified and allocated codewords according to the frequencies with which they occur. These include the 'cumulative reference library' concept of Notley and similar approaches by Mayne and James and Rubin. Rubin introduces different algorithms for the recognition and encoding of input groups and, presents the results of empirical analyses to compare efficiencies, based on the required storage space, for three illustrative files. These methods require to store diction-

---

150 THE COMPUTER JOURNAL, VOL. 30, NO. 2, 1987
2.4 Arithmetic coding

There has been significant work on other than block codes such as the arithmetic coding techniques\(^5\) \(,\) \(\)\(^9\) \(,\) \(\)\(^10\) and similar methods\(^1\) \(,\) \(\)\(^2\) \(,\) \(\)\(^3\) \(,\) \(\)\(^4\) which can produce high compression efficiency and efficient encoding and decoding algorithms. The technique (see Ref. \(5\)), encodes source symbols into successive smaller partitions of the number line \(0\) to \(1\), and represents them as binary fractions if a binary code alphabet is used. The decoding process recovers the data by magnitude comparisons on the code sequence. A variant of the technique, \(P\)-based arithmetic codes, avoids problems of discrimination by using integer partitions, but they do not guarantee to use the code space efficiently. The idea underlying the method to be discussed in this paper has similarity to \(P\)-based arithmetic codes but retains the facility for instantaneous encoding and decoding, using all but a negligible amount of the code space available.

2.5 Objectives

The objectives of the method to be discussed are as follows.

(a) To retain as much of the true structure of the source as possible during the derivation of the encoding transformation.

(b) To select variable-length messages having probabilities as evenly distributed as possible, providing for efficient coding.

(c) To permit the code alphabet to be preselected in order to suit the application. Conceptually, the alphabet will be a set of single symbols but in reality may represent any set of convenient codewords.

(d) To provide for instantaneous encoding and decoding.

(e) To specify the encoding transformation as a finite-state model, economical in storage space and guaranteeing instantaneous encoding. The model may be simplified and manipulated by well-proven techniques and provides a formal model from which encoding and decoding devices may be designed systematically.

(f) The encoding transformation is to be optimum in the sense that no other encoding transformation, of the same type, could give a more efficient use of the code alphabet.

3. STATISTICAL PROPERTIES OF THE SOURCE

3.1 Formal representation

Consider a source, with an alphabet of \(n\) symbols, for which the generation of a symbol, at any instant in time, is dependent on the preceding sequence of symbols. The preceding sequences of symbols are represented by \(h\) states, which define classes of previous histories. For a zero-memory source \(h = 1\). Define such a Markov source \(M_{h,n}\) to be a probabilistic finite state machine the statistical behaviour of which is defined by the set of \(h \times n\) quadruples, \([Y_i, a_i, T_{ij}, P(T_{ij})]\) and which can be represented as in Table 1, where:

\(a_i\) is the source alphabet, \(1 \leq i \leq n\),

\(Y_i\) is the state set, \(1 \leq i \leq h\) and \(Y_i\) is the start state,

\(T_{ij} \in \{Y_1, \ldots, Y_h\}\) is the next state,

\((Y_i, a_i) \rightarrow T_{ij}\) is the state transformation function,

\(P(T_{ij})\) is the probability that the symbol \(a_i\) is generated when in state \(Y_i\).

Table 1. Specification of a Markov source

<table>
<thead>
<tr>
<th>State</th>
<th>Next state</th>
<th>Probability</th>
<th>State probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Y_1)</td>
<td>(T_1, \ldots, T_n)</td>
<td>(P(T_{11}) \ldots P(T_{1n}))</td>
<td>(P(Y_1))</td>
</tr>
<tr>
<td>(Y_2)</td>
<td>(T_1, \ldots, T_n)</td>
<td>(P(T_{21}) \ldots P(T_{2n}))</td>
<td>(P(Y_2))</td>
</tr>
<tr>
<td>(\vdots)</td>
<td>(\vdots)</td>
<td>(\vdots)</td>
<td>(\vdots)</td>
</tr>
<tr>
<td>(Y_h)</td>
<td>(T_1, \ldots, T_n)</td>
<td>(P(T_{h1}) \ldots P(T_{hn}))</td>
<td>(P(Y_h))</td>
</tr>
</tbody>
</table>

The set of probabilities must satisfy \(\sum_{j=1}^{n} P(T_{ij}) = 1\) for \(1 \leq i \leq h\), which expresses the requirement that outgoing transitions from a state \(Y_i\) are exhaustive and their probabilities must sum to unity. The set of equations, \(P(Y_i) = \sum P(T_{ij}) \times P(T_{ij})\), summation for \(T_{ij} = Y_j\), \(2 \leq r \leq h\) (the aggregate probability of moving into a state must equate to the probability of being in that state) and \(\sum_{j=1}^{n} P(T_{ij}) = 1\) (the states \(Y_i\) are exhaustive and their probabilities must sum to unity), must be well conditioned.

3.2 Examples of information sources

Two theoretical models of sources will be considered to illustrate and compare methods of compacting data.

(a) Source A. Table 2 illustrates a zero-memory binary source for which the probabilities of generating symbols are \(P(0, 1) = (0.25, 0.75)\). The entropy is 0.8113 bits per source symbol. Symbols occur independently; since the source is zero-memory, higher-order extensions are obtained by simple multiplication and entropies are linearly related. For the first extension \(P(00, 01, 10, 11) = (0.625, 0.1875, 0.1875, 0.5625)\) and the entropy is 1.6226 bits per source symbols.

Table 2. Specification for Source A

<table>
<thead>
<tr>
<th>State</th>
<th>Next state</th>
<th>Probability</th>
<th>State probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\rightarrow 1)</td>
<td>(0, 1)</td>
<td>(0.25, 0.75)</td>
<td>(1.0)</td>
</tr>
</tbody>
</table>

(b) Source B. Table 3 illustrates a source with a four-symbol alphabet \([a, b, c, d]\). The source is restricted and a sequence from the set \([aa, ab, bb, ba, cb, ca, da, db, bc, cc]\) is never permitted to occur. Otherwise if there is a choice of next symbol then symbols occur with equal probability. This is a Markov source which demonstrates features occurring in both natural and formal languages. The entropy is 0.560 bits/source symbol. It is seen that when interaction between symbols is ignored the probabilities of the corresponding zero-memory source, the first adjoint, are \(P(a,b,c,d) = (0.16, 0.32, 0.28, 0.24)\).
Table 3. Specification for Markov Source B

<table>
<thead>
<tr>
<th>State</th>
<th>Next state</th>
<th>Probability</th>
<th>State probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b</td>
<td>c</td>
<td>d</td>
</tr>
<tr>
<td>A</td>
<td>—</td>
<td>—</td>
<td>B</td>
</tr>
<tr>
<td>B</td>
<td>C</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>C</td>
<td>D</td>
<td>A</td>
<td>—</td>
</tr>
<tr>
<td>D</td>
<td>C</td>
<td>E</td>
<td>—</td>
</tr>
<tr>
<td>E</td>
<td>A</td>
<td>B</td>
<td>—</td>
</tr>
<tr>
<td>F</td>
<td>D</td>
<td>C</td>
<td>G</td>
</tr>
<tr>
<td>G</td>
<td>—</td>
<td>A</td>
<td>B</td>
</tr>
</tbody>
</table>

By ignoring interaction between groups of symbols (the higher adjoints) source-extension probabilities may be derived. For the second adjoint: \( P(ab ac ba bc cc cd db) = (0.08 0.08 0.16 0.47 0.23 0.24) \) and for the third adjoint: \( P(aba abc acc acc dbac edbc cdce dbac) = (0.04 0.04 0.04 0.08 0.08 0.16 0.47 0.23 0.12 0.12) \). The entropies of the adjoints measured in bits/source symbol are: 1st 1.957, 2nd 1.310, 3rd 1.060, 4th 0.933, 5th 0.859, 6th 0.809.

By ignoring some or all of the interaction between symbols the information-communicating capability of the source is significantly overestimated. In order to use the model for simulation of the generation of symbols from the source a starting state (or sequence of states) is needed which is the average over all states of the source. This gives rise to a non-ergodic source in which the entry states (states F and G of Table 3) are never entered more than once. Over a long sequence their effect is minimal and the behaviour of the source approximates to the ergodic Markov source defined by states A, B, C, D and E.

4. STATISTICAL SPECIFICATION OF AN ENCODER

An encoder receives sequences of symbols, as inputs, from an alphabet of \( n \) symbols, the statistical properties of which are specified by a Markov source which emits a symbol on each transition to a new state. An encoder can be specified in a similar manner, except that the encoder accepts a symbol, as input, on each transition to a new state. Code symbols are output from the encoder from an alphabet of \( R \) symbols.

4.1 Definition

A probabilistic encoder, \( E_{m,n} \), is a probabilistic finite state recogniser, the statistical behaviour being defined by the set of \( m \times n \) quintuples \([X_i, a_j, S_{ij}, P(S_{ij}), z_i]\), which can be represented as in Table 4 where:

- \( a_j \) is the source alphabet, \( 1 \leq j \leq n \)
- \( X_i \) is a transition state, \( 1 \leq i \leq m \) and \( X_1 \) is the start state.
- \( X_i \) are accept states, \( m < i \leq mn+1 \)
- \( Z_i \) is the type of Markov state, \( Z_i = Y_i \)
- \( S_{ij} \in [X_1, \ldots, X_m] \) is the next state, \( (X_i, a_j) \rightarrow S_{ij} \) is the state transformation function, \( P(S_{ij}) \) is the probability of entering state \( S_{ij} \).

The set of probabilities must satisfy \( P(X_i) = \Sigma P_{x_{i1}} P(S_{ij}) \) for \( 1 \leq i \leq m \), once a state is entered it is certain to receive an input symbol and move into a successor state. \( P(X_1) = 1 \) since the encoder is assumed to start in state \( X_1 \).

4.2 Design of encoder and decoder

To specify the encoding transformation, it is necessary to refer only the next state table, which can be appended to the file to be transmitted. Replace source symbols \( a_j \) and states \( X_i \) by ordinal positions \( i \) and \( j \), to give an \( m \times n \) table of integers \( S[i,j] \) in the range 2 to \((m \times n+1)\). A state such that \( i > m \) represents an Accept state. It is required to output a codeword (ordinal position \( i-m-1 \)), then return to state 1. The encoding process can be implemented by an algorithm of the form:

Set \( i \) to 1
Repeat input (source symbol), replace \( i \) by \( S[i,j] \)
if \( i > m \) then output (code symbol \( i-m+1 \))
replace \( i \) by 1
Until end of data source.

A table of \( R \) variable-length source strings corresponding to \( R \) codewords can easily be derived from the finite state model. The decoding process can then be implemented by an algorithm of the form:

Repeat input (code symbol \( k \))
output (string from \( k \)th table entry)
Until end of Code data.

It may be convenient to replace all Accept states by state 1 and set up an \( m \times n \) entry table of output codes, or signals to call actions to compute the output codes. Methods are available to minimise the number of states and hence the size of the model. The code symbols can be ordered in a way suitable to the application and accept-state labels may be allocated to assist in simplifying the finite state model. Examples of this are presented in working papers.6

For Huffman coding, the length of each path from start state to accept state is constant, equal to the order of the adjoint or extension used. Codewords associated with accept states may be of variable length. Table 5 illustrates this representation for the third adjoint extension of Source B, in which [a:b ac:acc ab:bac bcd ced dba] correspond to accept states (13 14 15 16 17 18 19 20 21 22 23) and which are coded as binary codewords [1110 1111 00000 00001 0001 0010 0101 0111 1110] respectively. The average length of codeword is 1.071 bits per source symbol, to be compared with the entropy of the third adjoint which is 1.060 bits.

In contrast, in the method to be discussed, source strings correspond to variable-length paths, and outputs are from the set of available code symbols.
Table 5. Finite-state recogniser for Huffman coding

<table>
<thead>
<tr>
<th>State</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>Probability</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>State</th>
<th>State type</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>0.16</td>
<td>0.32</td>
<td>0.28</td>
<td>0.24</td>
<td>F</td>
<td>1.0</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>6</td>
<td>7</td>
<td>1</td>
<td></td>
<td>0.00</td>
<td>0.08</td>
<td>0.08</td>
<td>0.00</td>
<td>D</td>
<td>0.16</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>8</td>
<td>9</td>
<td>1</td>
<td></td>
<td>0.16</td>
<td>0.00</td>
<td>0.16</td>
<td>0.00</td>
<td>C</td>
<td>0.32</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>10</td>
<td>11</td>
<td></td>
<td></td>
<td>0.00</td>
<td>0.00</td>
<td>0.047</td>
<td>0.233</td>
<td>G</td>
<td>0.28</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>12</td>
<td></td>
<td></td>
<td></td>
<td>0.00</td>
<td>0.24</td>
<td>0.00</td>
<td>0.00</td>
<td>B</td>
<td>0.24</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>13</td>
<td>14</td>
<td></td>
<td></td>
<td>0.00</td>
<td>0.00</td>
<td>0.04</td>
<td>0.00</td>
<td>C</td>
<td>0.08</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td>15</td>
<td>16</td>
<td></td>
<td>0.00</td>
<td>0.08</td>
<td>0.08</td>
<td>0.00</td>
<td>E</td>
<td>0.08</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>17</td>
<td>18</td>
<td></td>
<td></td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.16</td>
<td>D</td>
<td>0.16</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td>19</td>
<td></td>
<td></td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.047</td>
<td>A</td>
<td>0.16</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>21</td>
<td>20</td>
<td></td>
<td></td>
<td>0.00</td>
<td>0.233</td>
<td>0.00</td>
<td>0.00</td>
<td>B</td>
<td>0.233</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
<td>23</td>
<td></td>
<td></td>
<td>0.00</td>
<td>0.00</td>
<td>0.12</td>
<td>0.00</td>
<td>C</td>
<td>0.24</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
<td></td>
<td>23</td>
<td></td>
<td>0.12</td>
<td>0.00</td>
<td>0.12</td>
<td>0.00</td>
<td>C</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

5. A VARIABLE-LENGTH MESSAGE APPROACH

We consider a source, with statistical characteristics specified by the probabilistic finite-state machine \( M_{b,n} \). A set of variable length messages from the source will be selected such that the code is instantaneous, i.e., no message will be the prefix of any other message (Ref. 1, p. 50). It will be assumed, initially, that the design is capable of encoding any sequence of source symbols, even though the probability of some sequences may have been estimated to be zero.

Given a number, \( m \), of fixed-length codewords available, a probabilistic finite-state recogniser, \( F_{m,n} \), will be derived to represent a set, not greater than the number of codewords, of variable-length source messages, with the objective that the average message length is maximum. The encoding transformation will be specified by the finite-state recogniser, which will also serve as a model from which encoding and decoding devices may be designed.

**Theorem 1**

\[(R - n - 1) \leq \text{number of Accept states} < (R - 2)\] in order that the maximum number of available code symbols is used.

**Proof**

\[m \times n \leq (m-1) + R \quad \text{and} \quad (m+1) \times n > R + m, \quad \text{i.e.,} \quad (R-n)/\quad (n-1) < m < (R-1)/(n-1), \quad \text{since next-state entries are unique.} \] The number of Accept states is \( m \times (n-1) - 1 \).

Therefore \( (R - n - 1) < \text{number of Accept states} \leq (R - 2) \).

**Theorem 2**

\[\sum_{i=1}^{n} P(S_{ij}) = P(X_i) \quad \text{for} \quad 1 \leq i \leq m \quad \text{and} \quad P(X_1) = 1.\]

**Proof**

The encoding process always starts at state \( X_1 \) which corresponds to the first state \( Z_1 \) of the specification of the Markov source. Then \( P(X_1) = 1 \) and \( P(S_{ij}) \) can be generated iteratively: \( P(S_{ij}) = P(X_i) \times P(T_{kj}) \) where state type \( Z_t = Y_k \).

\[\sum_{j=1}^{n} P(S_{ij}) = \sum_{j=1}^{n} P(X_i) \times P(T_{kj}) = P(X_1) \times \sum_{j=1}^{n} P(T_{kj}) = P(X_i).\]

The average length of the source-symbol sequence corresponding to a code symbol is given by \( L_m = \Sigma P(S_{ij}) \times \text{(path length} X_i \text{to} S_{ij}) \) where summation is over all \( S_{ij} = X_i \text{for} \, m < i \leq mn + 1 \). It can be easily shown that \( L_m = \Sigma_{S_{ij}} P(S_{ij}) \text{ since} S_{ij} \text{ are unique and} \) \( P(X_i) = \Sigma_{j=1}^{n} P(S_{ij}) \) for \( 1 \leq i \leq m \). The efficiency of the encoding transformation, where the source entropy \( H \) is in \( m \)-units, is defined by \( E_{m,n} = H/CSR = H \times L_m \).

**Theorem 3**

\( L_m = \Sigma_{S_{ij}} P(S_{ij}) \) is maximum for \( P(S_{ij}) \geq P(S_{ij}') \)

where \( S_{ij} = X_i \quad 1 \leq i \leq m \) (transition states) and \( S_{ij}' = X_a \quad m < a \leq mn + 1 \) (accept states).

**Proof**

Suppose that, for some encoder with maximum average length \( L_m \), there are states \( S_{ij} = X_i \) and \( S_{ij}' = X_a \) such that \( P(X_i) < P(X_a) \) and all 1-successors \( S_{ij} \) are themselves accept states. If a 1-successor \( S_{ij} \) is itself a transition state then \( X_i \) can be replaced by \( S_{ij} \) in the supposition since \( P(S_{ij}) < P(X_i) \). Evaluate the probabilities of 1-successors for state \( X_a \) and insert 1-successors \( S_{ij} \) for \( 1 \leq j \leq n \) in the encoder representation. The average length \( L \) is then \( L_m + \Sigma_{S_{ij}} P(S_{ij}) \). In the encoder representation replace \( S_{ij} \) by \( X_a \) and delete all successors \( S_{ij} \) \( 1 \leq j \leq n \). Then

\[L = L_m + \sum_{j=1}^{n} P(S_{ij}) - \sum_{j=1}^{n} P(S_{ij}) = L_m + P(X_a) - P(X_1) > L_m.\]

But \( L_m \) is the maximum average length. Hence the supposition that there exists \( S_{ij} = X_i \) and \( S_{ij}' = X_a \) such that \( P(X_i) < P(X_a) \) cannot hold.

**Definition**

A probabilistic encoder \( F_{m,n} \) with \( m \) transition states, such that \( P(X_a) \geq P(X_b) \) for all \( 1 \leq a \leq m \) and \( m + 1 \leq b \leq mn + 1 \) is defined to be an Optimum Encoder \( E_{m,n} \).

The number of accept states, i.e., code symbols, for...
an optimum encoder is \( mn - m + 1 \). There is no other probabilistic encoder \( E_{m,n} \) which has a shorter average source-symbol length per code symbol. The efficiency of the encoding transformation, defined by \( E_{m,n} \) is \( H \times L_m \) where \( H \) is the source entropy in \( m \)-units. Although for a given number of codewords the encoding transformation defined by \( E_{m,n} \) has optimal efficiency, it is not necessarily true that the efficiency increases as \( m \), the number of available codewords, increases.

**Theorem 4**
If \( E_{m,n} \) is an optimum encoder then

\[
\text{Efficiency } (E_{m+n-1,n}) < \text{Efficiency } (E_{m,n})
\]

iff \( P(X_i) = (\log(m+n-1))/\log(m)-1 \times L_m \) for all \( m < t \leq mn+1 \).

**Proof**

Source entropy = \( H/\log(m) \) for the optimum encoder \( E_{m,n} \) where \( log(m) \) and the \( H \) units are to the same base. CSR = \( 1/L_m \) and Efficiency = \( (H \times L_m)/\log(m) \).

\( L_{m+n-1} = L_m + P(X_i) \) for the optimum encoder \( E_{m+n-1,n} \) where, for \( m < t \leq mn+1 \) and for all \( m < b \leq mn+1 \), \( P(X_i) \geq P(X_b) \).

Efficiency \( (E_{m+n-1,n}) = \frac{H \times (L_m + P(X_i))}{\log(m+n-1)} \)

Hence

Efficiency \( (E_{m+n-1,n}) < \text{Efficiency } (E_{m,n}) \)

iff \( (H \times (L_m + P(X_i)))/\log(m+n-1) < (H \times L_m)/\log(m) \)
i.e. \( P(X_i) < \log(m+n-1)/\log(m)-1 \times L_m \)

for \( m < t \leq mn+1 \).

### 6. Development of an Algorithm to Derive an Optimum Encoder

Given a Markov source, \( M_{h,n} \), it is required to derive an optimum encoder for a source alphabet of \( n \) symbols and a code alphabet of \( R \) symbols.

#### 6.1 Outline of development

Redefine all accept states of \( E_{k,n} \) as \( X_i \).

**Precondition**

Given a probabilistic encoder \( E_{1,n} \) with length \( L_1 \) equal to \( 1 \) and with \( C_i \) accept states, \( i \leq C_i \leq n \), for which the single transition states the start of the Markov source specification.

**Invariant**

\( E_{k,n} \) of length \( L_k \) with \( C_k \) accept states for \( 1 \leq k \leq m \) is an optimum encoder, \( S_{ab} = X_k \) with probability \( Q \).

**Postcondition**

To derive an optimum encoder, \( E_{m,n} \) such that \( (R-n) < m(n-1) \leq R-1 \), i.e. \( R-n-1 < C_m \leq R \).

**Algorithm**

 Initialise \( E_{1,n} \) [Set first state to that of \( M_{h,n} \)]

**Repeat** Locate \( S_{ab} \) [of state type \( t \) with probability \( Q \)]

satisfying criteria for Optimum Encoder \( E_{k+1,n} \);

Expand to \( E_{k+1,n} \);

[Evaluate state \( k+1 \); Replace \( k \) by \( k+1 \)]

Until \( C_k > R \);

Reduce to \( E_{k-1,n} \); [Disregard state \( k \)]

### 6.2 Refinements of outline algorithm

Refinement of Initialise \( E_{1,n} \)

Set \( k \) to \( 1 \); Set \( C_1 \) to \( n \); Set \( Q = L_1 \) to \( 1 \); Set \( Z_1 \)
to \( Y_1 \);

For \( 1 < j < n \); Set \( S_{1j} \) to \( X_j \); Set \( P(S_{1j}) \) to \( P(T_{ij}) \);

[Use \( M_{h,n} \)] [*** If \( P(T_{ij}) = 0 \) then Replace \( C_j \) by \( C_j-1 \); ***]

Criteria to Locate \( S_{ab} \)

\( S_{ab} \) is an accept state (\( X_i \)) for \( 1 < a < k \), 1 < \( b < n \) such that \( P(S_{ab}) > P(S_{ab}) \) for all \( 1 < i < k \), 1 < \( j < n \).

Refinement of Expand to \( E_{k+1,n} \)

Replace \( k \) by \( k+1 \); Replace \( C_k \) by \( n-1 + C_k-1 \);

For \( 1 < j < n \); Set \( S_{kj} \) to be accept states;

Set \( P(S_{kj}) \) to \( Q \times P(T_{ij}) \); [Use \( M_{h,n} \)]

[*** If \( P(T_{ij}) = 0 \) then Replace \( C_j \) by \( C_k-1 \); ***]

Replace \( S_{ab} \) by \( X_k \); Set \( Z_k \) to \( t \); Replace \( L_k \) by \( Q + L_k-1 \);

Refinement of Reduce to \( E_{k-1,n} \);

Set \( m \) to \( k-1 \); Replace \( S_{ab} \) by accept-state \( X_j \);

Noting that \( P(T_{ij}) < 1 \), for all \( t \), then \( P(S_{kj}) < P(X_k) \).

It is self-evident that the required invariant condition holds and that the required postcondition is achieved. The optimum encoder \( E_{m,n} \) is then specified as required. (The statements marked *** are only required if input sequences with zero probability are not to be allocated code symbols from the code alphabet.)

### 6.3 Zero probability source sequences

When specifying the statistics of a source it may be known that a particular sequence can never arise and that the zero probability associated with it is absolute. Alternatively a zero probability may result from an estimate, which represents a ‘very rare’ occurrence of a sequence. Three approaches may be adopted to deal with zero probability sequences in the design of a code.

(a) Allocate a codeword to each conceivable sequence of source symbols, even though the probability of a sequence may be zero, and define for each a corresponding state transition in the resultant finite-state recogniser. This may result in loss of efficiency, since all the available code space may not be used.

For example, source \( B \) is to be encoded into eight codewords.

Encode:

\[[a \ b \ b \ b \ c \ d \ c \ d \ c \ d] \rightarrow [c1 \ c2 \ c3 \ c4 \ c5 \ c6 \ c7 \ c8]\]

\( P(a \ b \ b \ b \ c \ d \ c \ d \ c \ d) = (0.28 \times 0.16 \times 0.00 \times 0.16 \times 0.24 \times 0.24) \)

Three of the codewords are not used, two which can never arise, (\( c3, c4 \)) and one \( (c8) \) because an instantaneous code is required and the number of extra codewords introduced at each stage constrains the total used.

(b) Reserve one codeword to represent all source sequences with zero probability. Should such a sequence
subsequently arise in practice the receiver is notified that a rare source sequence has been transmitted but will have no means of distinguishing between the possible alternatives without reference to other external information – such as knowledge of the context. The unused code space could be significantly reduced.

(c) For a source sequence with zero probability, no codeword is allocated and the corresponding state transition is denoted as an arbitrary ‘don’t care’ transition. In this way all the code space can be utilised and the ‘don’t care’ entries give scope for simplification of the finite state recogniser defining the transformation. Should such a sequence subsequently arise in practice it will be encoded and decoded incorrectly and will not be distinguishable from a sequence for which a codeword has been allocated. This approach can be implemented by including those statements marked \*$*$ in the algorithm given earlier.

It should be noted that, although the elimination of zero probability codewords can improve the efficiency of encoding, it cannot be guaranteed that the final encoding transformation is the optimum possible. The proofs of theorems given earlier assume that codewords are allocated to all source sequences.

For example, consider an encoding design for code B, with nine codewords available. After six steps of the algorithm the recogniser has seven states, corresponding to the sequences: (a ba bc cc cdb cd bca dbe) with probabilities (0.16 0.16 0.16 0.05 0.116 0.116 0.12 0.12) and average length \(L = 2.55\). There are three alternative sequences with highest probability 0.16 which can be extended:

\[
\begin{align*}
a &\rightarrow P(\text{aa ab ac ad}) = (0.08 0.08 0.08) , 9 \text{ codewords} \\
b &\rightarrow P(\text{ba bab bac bad}) = (0.08 0.08 0.08) , 9 \text{ codewords} \\
c &\rightarrow P(\text{bc bcb bcc bcd}) = (0.00 0.016) , 8 \text{ codewords} \\
\end{align*}
\]

In all cases the average length is increased by 0.16, but in the third case further expansion is possible since the number of sequences has not been increased. It is clear that the difficulty only arises when there is a choice of extending equally probable sequences, at the final stage of the algorithm, when the codewords to be allocated to source sequences, with non-zero probability, equal or exceeds the number available. As the number of available codewords is increased, this end effect becomes negligible.

7. RESULTS AND COMMENTS

Encoding transformations have been produced, using the variable-length message encoding algorithm as described in the previous section. In Tables 6, 7 and 8:

- \(m\) is number of symbols in code alphabet (codewords);
- \(L\) is average length of message from source alphabet;
- \(H\) is source entropy, in \(m\)-units;
- \(E\) is efficiency of encoding transformation, \(E = 100 \times L \times H\);
- \(k\) is number of states in finite state recogniser;
- \(p\) is maximum of accepting state probabilities;
- \(D\) is \(L \times (\log (m + n - 1)/\log (m) - 1)\).

7.1 Results for source A

Table 6 shows how the average source message length \(L\), and the efficiency \(E\), changes for varying code alphabet sizes \(m\), where \(m\) takes the values 2, 4, 8, 16, 32, 64, 128 and 256 corresponding to 1, 2, 3, 4, 5, 6, 7, 8-bit codewords from a binary alphabet. The number of states \(k\) in the unsimplified finite-state recogniser is \((m - 1)\). There are two source symbols associated with state 1. For every extra state introduced one source sequence is extended and another introduced, requiring a further codeword, because all sequences are possible. It is seen that \(L\) approaches the inverse of the entropy as the size of the code alphabet increases, the improvement being less marked for larger code alphabets.

Table 7 exemplifies the effect of Theorem 4 and shows that efficiency does not necessarily improve uniformly as the size of the code alphabet increases. In the example, as the number of codewords increases from 6 to 7, the probability \(p\) is less than \(D\), i.e. \((L + p)/\log (m + 1) < L/\log (m)\) and the efficiency decreases from 95.75% to 95.39%.

For the case \(m = 16\), the variable-length source strings of Source A which are encoded and allocated to the 16 codewords are: [00, 010, 0110, 0111, 100, 1010, 1011, 1100, 11010, 11011, 1110, 11110, 111110 and 111111].
Table 8. Variable-length message encoding for source B

<table>
<thead>
<tr>
<th>m</th>
<th>4</th>
<th>8</th>
<th>16</th>
<th>32</th>
<th>64</th>
<th>128</th>
<th>256</th>
</tr>
</thead>
<tbody>
<tr>
<td>L</td>
<td>1.000</td>
<td>1.320</td>
<td>2.080</td>
<td>3.027</td>
<td>4.373</td>
<td>5.737</td>
<td>7.281</td>
</tr>
<tr>
<td>H</td>
<td>0.2800</td>
<td>0.1867</td>
<td>0.1400</td>
<td>0.1120</td>
<td>0.0933</td>
<td>0.0800</td>
<td>0.0700</td>
</tr>
<tr>
<td>E</td>
<td>28.0</td>
<td>26.3</td>
<td>29.1</td>
<td>34.2</td>
<td>40.8</td>
<td>46.0</td>
<td>51.0</td>
</tr>
<tr>
<td>k</td>
<td>1</td>
<td>2</td>
<td>5</td>
<td>10</td>
<td>21</td>
<td>42</td>
<td>85</td>
</tr>
</tbody>
</table>

(a) All codewords allocated

(b) Zero probability codewords not allocated

<table>
<thead>
<tr>
<th>L</th>
<th>1.000</th>
<th>2.867</th>
<th>4.373</th>
<th>5.903</th>
<th>7.884</th>
<th>9.652</th>
<th>11.418</th>
</tr>
</thead>
<tbody>
<tr>
<td>H</td>
<td>0.2800</td>
<td>0.1867</td>
<td>0.1400</td>
<td>0.1120</td>
<td>0.0933</td>
<td>0.0800</td>
<td>0.0700</td>
</tr>
<tr>
<td>E</td>
<td>28.0</td>
<td>53.5</td>
<td>61.2</td>
<td>66.6</td>
<td>73.6</td>
<td>77.2</td>
<td>79.9</td>
</tr>
<tr>
<td>k</td>
<td>1</td>
<td>9</td>
<td>21</td>
<td>47</td>
<td>109</td>
<td>222</td>
<td>450</td>
</tr>
</tbody>
</table>

7.2 Results for source B

Table 8 shows how the average source-message length L, and the efficiency E, change for varying code alphabet sizes m, with values 4, 8, 16, 32, 64, 128 and 256 which correspond to 2, 3, 4, 5, 6, 7 and 8-bit codewords from a binary alphabet. In the first case codewords are allocated to all symbol sequences, including those with zero probability. The number of states is the integer part of (m − 1)/3; for each extra state three additional symbol sequences are introduced.

In the second case, a codeword is not allocated to any symbol sequence with zero probability on the assumption that it can never occur. The number of states is considerably increased as, in some stages of the algorithm, no further source sequence is introduced when adding an extra state (e.g. Markov type A or B). As is expected, there is significant improvement in efficiency as restrictions placed on the source language are taken into account and the number of codewords is increased.

For the case m = 16, the variable-length source strings of source B which are encoded and allocated to the 16 codewords are: [ab, ac, bab, bace, baced, bdaba, bcdbe, cc, cdbab, cdbac, cbdcdb, cbdcba, dbab, dbac, dbcdba, and dbdcbe].

REFERENCES