Data Compression Using Adaptive Coding and Partial String Matching

JOHN G. CLEARY AND IAN H. WITTEN, MEMBER, IEEE

Abstract—**The recently developed technique of arithmetic coding, in conjunction with a Markov model of the source, is a powerful method of data compression in situations where a linear treatment is inappropriate. Adaptive coding allows the model to be constructed dynamically by both encoder and decoder during the course of the transmission, and has been shown to incur a smaller coding overhead than explicit transmission of the model's statistics. But there is a basic conflict between the desire to use high-order Markov models and the need to have them formed quickly as the initial part of the message is sent. This paper describes how the conflict can be resolved with partial string matching, and reports experimental results which show that mixed-case English text can be coded in as little as 2.2 bits/character with no prior knowledge of the source.**

I. INTRODUCTION

RECENT approaches to data compression have split the problem into two parts: modeling the statistics of the source and transmitting a particular message generated by that source in a small number of bits [19]. For the first part, Markov modeling is generally employed, although the use of language-dependent word dictionaries in data compression has also been explored [21]. In either case the problem of transmitting the model must be faced. The usual procedure is to arrange that when the transmission system is set up, both encoder and decoder share a general model of the sorts of messages that will be sent. The model could take the form of a table of letter or diagram frequencies. Alternatively, one could extract appropriate statistics from the message itself. This involves a preliminary scan of the message by the encoder and a preamble to the transmission which informs the decoder of the model statisitics. In spite of this overhead, significant improvement can be obtained over conventional compression techniques.

The second part, transmitting a message generated by the source in a small number of bits, is easier. Conceptually, one can simply enumerate all messages which can be generated by the model and allocate a part of the code space to each whose size depends on the message probability. This procedure of enumerative coding [6] unfortunately becomes impractical for models of any complexity. However, the recent invention of arithmetic coding [15] has provided a method which is guaranteed to transmit a message in a number of bits which can be made arbitrarily close to its entropy with respect to the model which is used. The method can be thought of as a generalization of Huffman coding which performs optimally even when the statistics do not have convenient power-of-two relationships to each other. It has been shown to be equivalent to enumerative encoding [5], [18], [19], and gives the same coding efficiency.

Paper approved by the Editor for Data Communication Systems of the IEEE Communications Society for publication without oral presentation. Manuscript received May 7, 1982; revised May 22, 1983. This work was supported by the Natural Sciences and Engineering Research Council of Canada.

The authors are with the Department of Computer Science, University of Calgary, Calgary, Alta., Canada T2N 1N4.

There are obvious disadvantages in having the encoder and decoder share a fixed model which governs the coding of all messages. While it may be appropriate in some tightly defined circumstances, such as special-purpose machines for facsimile transmission of documents [9], it will not work well for a variety of different types of message. For example, imagine an encoder embedded in a general-purpose modem or a computer disk channel. The most appropriate model to use for such general applications may be one of standard mixed-case English text. But the system may have to encode long sequences of upper-case-only text, or program text, or formatted bibliography files—all with statistics quite different from those of the model.

There is clearly a case for basing the model on the statistics of the message which is currently being transmitted. But to do so seems to require a two-pass approach, with a first pass through the message to acquire the statistics and a second for actual transmission. This procedure is quite unsuitable for many applications. Usually, one wishes to begin sending the message before the end of it has been seen. The obvious solution is to arrange that both sender and receiver adapt the model dynamically to the message statistics as the transmission proceeds. This is called "adaptive coding" [12]. It has been shown theoretically [5] that for some models, adaptive coding is never significantly worse than a two-pass approach and can be significantly better. This paper verifies these results in practice for adaptive coding using a Markov model of the source.

But even with adaptive coding, there is still a problem in the initial part of the message because not enough statistical information has been gained for efficient coding. On the one hand, one wishes to use a high-order Markov model to provide as much data compression as possible once appropriate statistics have been gathered. But it takes longer to gather the statistics for a high-order model. So, on the other hand, one wishes to use a low-order model to accelerate the acquisition of frequency counts so that efficient coding can begin sooner in the message. Our solution is to use a "partial match" strategy, where a high-order model is formed but used for lower order predictions in cases when high-order ones are not yet available.

The structure of this paper is as follows. The next section presents the coding method and the partial string match strategy which is used to gain good performance even early in the message. In Section III the results of some experiments with the coding scheme are presented. These experiments use a variety of different sorts of data for compression, and respectable—in some cases exceptionally good—performance is achieved with all of them. Finally, the resources required to use the method in practice are discussed. An appendix gives a formal definition of how character probabilities are estimated using partial string matching.

II. THE CODING METHOD

Arithmetic Coding

Arithmetic coding has been discussed recently by a number of authors [7], [10], [15], [19]. Imagine a sequence of

0090-6778/84/0400-0396$01.00

symbols $X_1X_2 \cdots X_N$ is to be encoded as another sequence $Y_1Y_2 \cdots Y_M$. After a sequence $X_1 \cdots X_{i-1}$, the possible values for X_i are ordered arbitrarily, say $w_1 \cdots w_m$, and assigned probabilties $p(w_1) \cdots p(w_m)$. The encoding Y is computed iteratively using only these probabilities at each step.

Arithmetic coding has some interesting properties which are important in what follows.

- The symbol probabilities $p(w_k)$ may be different at each step. Therefore, a wide range of algorithms can be used to compute the $p(w_k)$, independent of the arithmetic coding method itself.
- It is efficient and can be implemented using a small fixed number of finite arithmetic operations as each symbol of X is processed.
- The output code length is determined by the probabilities of the symbols X_i and can arbitrarily closely approximate the sum $\Sigma - \log p(X_i)$ if arithmetic of sufficient accuracy is used.

Adaptive Transmission of the Model

Arithmetic coding commonly uses fixed probabilities, contingent on the current context, which are derived from statistical analysis of text. However, our method derives the probabilities from the message itself. Furthermore, because encoding is done in a single pass through the message, the statistics are gathered from the *preceding* portion of the message only. Thus, they are continually changing with time as the transmission proceeds. Such an adaptive strategy has been used by Langdon and Rissanen [12] with fixed-order Markov models. Roberts [20] also discusses similar techniques applied to encoding and authorship identification of English text. Ziv and Lempel [24] have proposed a coding technique which involves the adaptive matching of variable length strings. Superficially, this technique appears to be very different from the "arithmetic coding plus adaptive model" approach pursued here. Nevertheless, it has been shown by Langdon [13] that there exists a scheme of this form which is equivalent to Ziv–Lempel coding.

It may at first seem difficult to implement a coding system based upon predictions whose probabilities are changing all the time. However, the problem is not so great as might be imagined, because usually, *nothing extra need be transmitted* to update the probabilities. After all, the decoder—if it is working properly—is seeing exactly the same message sequence as the encoder, and so it can update frequency counts just as easily as can the encoder. It is, of course, necessary that a character is encoded according to the old model, before the counts have been updated to take into account that occurrence of the character. Having encoded a character, the encoder updates its model. Having decoded it, the decoder updates its own model. Assuming error-free transmission—and this is an assumption that is made throughout this paper—the models will always agree, even though explicit details of the models are never transmitted. Appropriate error correction policies, or error detection and retransmission protocols, can be applied to the encoded data to make the probability of undetected errors arbitrarily small.

Markov Modeling with Partial String Matching

The coding scheme uses a Markov model which conditions the probability that a particular symbol will occur on the sequence of characters which immediately precede the symbol. The order of the Markov model, for which we use the symbol o, is the number of characters in the context used for prediction. For example, suppose an order-2 model is selected, and the current symbol sequence is "... #and#the#current#symbol#sequence#i." The next character, which will be denoted by φ, could be any member of the coding alphabet—we use 7 bit ASCII codes in this section. (Of course, the technique is not restricted to this alphabet.) In particular, characters such as ⟨*space*⟩ are significant. ⟨*space*⟩ is written here as "#" merely to enhance legibility.

Since the model is of order $o = 2$, the next character is predicted on the basis of occurrences of trigrams "# i φ" earlier in the message. A scan through an English dictionary will show that φ = "e," "h," "i," "j," "k," "q," "u," "w," "y," and "z" are unlikely in this context, while the high frequency of the word "is" will give φ = "s" a reasonably high probability in this context.

The coding scheme does not use a fixed order. If it did and the order o were large, then predictions would be infrequent until most of the $(o + 1)$-sequences which actually occur in the message had been seen. When a context occurs in which the following character has not been seen before—for example, on the first occurrence of "# i φ"—an escape mechanism is used to transmit the character identity.

Instead of using a fixed length context, both encoder and decoder recognize predictions on the basis of the longest-string match between the present context and previously seen ones. This creates no ambiguity because each sees the same message sequence. For example, when the character φ = "s" occurs in the context " # i φ" for the first time, the prediction will be based on the length-1 context "i φ". Thus, if the string "is" has occurred previously in the message, even without a preceding space (as in the word "history"), the coding of the character "s" will be based on this foreshortened context. In essence, both encoder and decoder use an *escape* mechanism to back down to the previous level. Then the character is encoded at this level, using the order-1 model which is implicit in the stored order-2 one.

If the string has not occurred previously, the context will be further shortened to the empty string. The encoder will use a second escape sequence to inform the decoder of this event. This will cause the character to be predicted on the basis of the order-0 model, that is, on its frequency so far in the message. If, however, the character has never been seen before, so that it is not predicted by the order-0 model, the escape mechanism is used a third time. The actual identity of the character is then transmitted using a probability of 1/128 for each.

Escape Probabilities

A case of great interest occurs when the encoder encounters a character in a context where it has never been seen before. For example, suppose φ = "s" occurs for the first time in the context "# i φ"—that is, the word "is," or indeed any other word which begins "is," has not occurred in the message so far. (This, of course, will happen quite frequently in the initial part of the message.) Then it is impossible for the encoder to encode it on the basis of its present model. Notice that we are talking here not just of the first occurrence of a particular character in the message, but of its first occurrence in each possible context.

For each context, one must allocate a probability to the event that a novel character occurs in that context. It is difficult to imagine a rationale for optimal choice of this probability. There has been extensive discussion of this problem by philosophers from at least the time of Kant. Pierce gives an outline of this early work in [16]. Some more modern solutions in the context of Markov models have been summarized by Roberts [20] who refers to this as the "zero frequency problem" (see also [17] and [1, p. 424]). Roberts also proposes his own solution (LOEMA) which takes weighted sums of the probability predictions of models of different orders. As noted by all these authors, in the absence of *a priori* knowledge, there seems to be no theoretical basis for choosing one solution over another.

We take a pragmatic approach to the problem, and have investigated the performance of two different expressions for the probability of a novel event. One motivation for the experiments described in the next section was to see if there is any clear choice between them in practice. Further experiments are needed to compare them with the other techniques referred to above.

The first technique estimates the probabilities for arithmetic coding by allocating one count to the possibility that some symbol will occur in a context in which it has not been seen before. Let $c(\varphi)$ denote the number of times that the symbol φ occurs in the context "# i φ" for each φ in the coding alphabet A (say, ASCII). Denote by C the total number of times that the context "# i" has been seen; that is, $C = \Sigma_{\varphi \in A} c(\varphi)$. Then, for the purpose of arithmetic coding, we estimate the probability of a symbol φ occurring in the same context to be

$$p(\varphi) = \frac{c(\varphi)}{1 + C}, \qquad c(\varphi) > 0.$$

The *escape probability* that some character occurs which is novel in that context, one for which $c(\varphi) = 0$, is therefore what remains after accounting for all seen characters:

$$1 - \sum_{\varphi \in A, c(\varphi) > 0} p(\varphi) = \frac{1}{1 + C}.$$

Let q be the number of characters that have occurred in that context, and a be the size of the coding alphabet A. Then there are $a - q$ characters that have not yet occurred in the context. We allocate to each novel character the overall coding probability

$$p(\varphi) = \frac{1}{1 + C} \cdot \frac{1}{a - q}, \qquad c(\varphi) = 0.$$

For convenience we call this technique method A.

The second technique, method B, classes a character in a particular context as novel unless it has already occurred *twice*. (This is motivated by the consideration that a once-off event may be an error or other anomaly, whereas an event that has occurred twice or more is likely to be repeated further.) The probability of an event which has occurred more than once in the context is then estimated to be

$$p(\varphi) = \frac{c(\varphi) - 1}{C}, \qquad c(\varphi) > 1.$$

The escape probability is therefore

$$1 - \sum_{\varphi \in A, c(\varphi) > 1} p(\varphi) = \frac{q}{C}.$$

We allocate to each novel character the overall coding probability

$$p(\varphi) = \frac{q}{C} \cdot \frac{1}{a - q}, \qquad c(\varphi) \leqslant 1.$$

A formal definition of the probability calculations used by partial string matching is given in the Appendix together with an example calculation of the probabilities. This includes an improvement whereby characters predicted by higher order models are neglected when calculating the probabilities of predictions by the lower order models.

III. Experimental Performance

The Sample Messages

The adaptive partial string match coding method has been tested on several different kinds of message. Table I summarizes the results, for a few different values of o, the order of the Markov model, using method A. Before discussing these results, however, we should say something about the kinds of message which were used.

TABLE I
SUCCESS OF THE PARTIAL STRING MATCH METHOD ON DIFFERENT KINDS OF MESSAGE

data	chars	bits/ char	bits/coded char o=0	o=1	o=2	o=3	o=4	o=9
1. English text	3,614	7	4.622	4.102	3.891	3.792	3.800	3.838
2. English text	551,623	7	4.555	3.663	2.939	2.384	2.192	--
3. English text	44,871	7	4.535	3.692	3.101	2.804	2.772	2.890
4. C source program	3,913	7	5.309	3.719	2.923	2.795	2.789	2.831
5. Bibliographic data	20,115	7	5.287	3.472	2.951	2.713	2.695	2.766
6. Numeric data in binary format	102,400	8	5.668	5.189	5.527	--	--	--
7. Binary program	21,505	8	6.024	5.171	4.931	4.902	4.914	4.950
8. Grey-scale picture as 8-bit pixel values	65,536	8	6.893	5.131	5.803	6.106	6.161	--
9. Grey-scale picture as 4-bit pixel values	65,536	4	3.870	1.970	1.923	1.943	1.991	--

Escapes calculated using Method A

For example, a 256-point transform with a sample rate of 8 kHz gives the 256 equally-spaced frequency components between 0 and 8 kHz that are shown in Table 4.2.

time domain sample number	time	frequency domain sample number	frequency
0	0 sec	0	0 Hz
1	125	1	31
2	250	2	62
3	375	3	94
4	500	4	125
.....			
.....			
.....			
254	31750	254	7938
255	31875 sec	255	7969 Hz

Table 4.2 Time domain and frequency domain samples for a 256-point DFT, with 8 kHz sampling
The top half of the frequency spectrum is of no interest, because it contains the complex conjugates of the bottom half (in reverse order), corresponding to frequencies greater than half the sampling frequency. Thus for a 30 Hz resolution in the frequency domain, 256 samples or a 32 msec stretch of speech, needs to be transformed. A common technique is to take overlapping periods in the time domain to give a new frequency spectrum every 16 msec. From the acoustic point of view this is a reasonable rate to re-compute the spectrum, for as noted above when discussing channel vocoders the rate of change in the spectrum is limited by the speed that the speaker can move his vocal organs, and anything between 10 and 25 msec is a reasonable figure for transmitting or storing the spectrum.

The DFT is a complex transform, and speech is a real signal. It is possible to do two DFT's at once by putting one time wave form into the real parts of the input and another into the imaginary parts. This destroys the DFT symmetry property, for it only holds for real inputs. But given the DFT of a complex sequence formed in this way, it is easy to separate out the DFT's of the two real time sequences. If the two time sequences are and , then the transform of the complex sequence

is

It follows that the complex conjugate of the aliased parts of the spectrum, in the upper frequency region, are

Fig. 1. Example text taken from data sample 3.

The first three samples are English text. All of them use upper and lower case characters in the normal way. Sample 1, the shortest, is an abstract of a technical paper. It includes some formatting controls as well as a ⟨*newline*⟩ character at the end of each line. Sample 2, the longest, is a complete 11-chapter book [23]. Notice that this sample contains over half a million characters. Prior to coding, we removed the formatting controls and mathematical expressions automatically, which left some rather anomalous gaps in the text. Tabular illustrations were not deleted. Fig. 1 shows a representative part of the text which includes a small table. Sample 3 is the first chapter from the book and, thus, forms a subsequence of sample 2: it is included to study how the coding efficiency is improved by exposing the coding scheme to a large, representative sample of text before transmitting the

actual message. The fourth sample is a computer program in source form, written in the language C [11], including some comments and ⟨*newline*⟩ characters. The fifth is a short extract from a bibliography file, which contains authors' names, titles, and reference details in a structured manner suitable for computer indexing and keyword retrieval [14]. These first five samples are all represented as ASCII text, requiring 7 bits/character.

The next three samples each use 8 bits/character. Sample 6 is a data file containing geophysical information in binary format. Sample 7 is a binary program, compiled from Pascal for the VAX 11/780 computer. Care was taken to remove the symbol table from it before coding so that no English-like text is included (apart from literal strings). Sample 8 is a grey-scale picture, histogram equalized and stored in raster order, with 256 grey levels. Finally, sample 9 is the same picture with pixels truncated to 4 bits (16 grey levels).

TABLE II
COMPARISON OF OPTIMUM COMPRESSION ACHIEVED BY ESCAPE TECHNIQUES

data	Method A o	Method A bits/char	Method B o	Method B bits/char	Improvement of B over A
1. English text	3	3.792	4	3.642	4.1%
2. English text	4	2.192	4	2.198	-0.3%
3. English text	4	2.772	5	2.746	0.9%
4. C source program	4	2.789	4	2.937	-5.0%
5. Bibliographic data	4	2.695	4	2.750	-2.0%
6. Numeric data in binary format	1	5.189	3	4.680	10.9%
7. Binary program	3	4.902	4	4.317	13.6%
8. Grey-scale picture as 8-bit pixel values	1	5.131	1	5.061	1.4%
9. Grey-scale picture as 4-bit pixel values	2	1.923	2	1.938	-0.7%

Performance of the Coding Scheme

Now we can examine the results of coding with method A which are presented in Table I. These are expressed in terms of bits/coded character. For example, the first line shows that a short English message can be coded in 3.792 bits/character, using the optimal value of $o = 3$. The penalty paid by choosing o too large is very small, however; for with $o = 9$ only 3.838 bits are needed—about 1.2 percent more. The optimal value of o grows slightly with the length of the message.

The piece of English text in sample 3 has an optimum at $o = 4$ (although this is not apparent from the table because the figure for $o = 5$ is not shown). For the text of sample 2 we were unable to carry the experiment beyond $o = 4$ for resource reasons. However, we have demonstrated that the method is able to code mixed-case English, including tables and rather arbitrary spaces, below 2.2 bits/character. And this is the *average* coding performance over the entire message—with both the encoder and decoder starting from scratch with no prior information at all about the likely statistics of the message. Although the table does not show it, the final 90 percent of this sample—half a million characters—was coded in 2.132 bits/character with $o = 4$. This indicates the performance which can be expected when the coding and decoding modules are primed with a short but representative sample of the kind of English used (55 000 characters in this case). Notice how much better it is than operating in unprimed mode for the short (45 000 character) text of sample 3—2.772 bits/character at the same value of $o = 4$.

It is interesting to compare the results for sample 4, the program in source form, with those for sample 1, which is English text of about the same length. For the lowest value of o, $o = 0$, the coding scheme does not perform as well with the program as it does with English. This is because of the abundance of unusual characters, like "{" and "*", in the program text, leading to a larger effective alphabet. (Anyone who has encountered the C language will assure you that it appears cryptic, especially at first.) However, performance improves with larger values of o, until at $o = 4$ (which is in fact the optimum for sample 4) the coded message occupies only 2.789 bits/character—73 percent of that for sample 1. This is because of the more structured form of a program: variables are all declared before they are used and are repeated relatively often, keywords are drawn from a relatively small set, the syntax constrains most operators to occur only after variables and not after keywords, and so on.

Another example of structured information is the bibliographic text file of sample 5. This contains formatted information together with free text in the form of titles, authors' names, and so on. At $o = 4$, coding with 2.695 bits/character is achieved, better than that obtained on the text file of similar size in sample 3.

Not surprisingly, a much smaller coding gain was obtained with binary data. The geophysical data of sample 6 can be coded in 5.189 bits/byte at the optimal value $o = 1$. This represents a reduction to 65 percent of the unencoded value of 8 bits/byte; much less that the reduction to 31–54 percent which was achieved for the samples of English. The compiled program of sample 7 takes 4.902 bits/byte, or 61 percent of the unencoded value. Presumably this is less "noisy" than the geophysical data, which would lead one to suspect that greater gains are possible for it. On the other hand, the coding in which machine-language programs are expressed has been carefully designed to eliminate redundancy.

The grey-scale pictures provide an interesting example. With 8 bit pixels, only 5.131 bits/pixel is achieved (64 percent) at the optimal value $o = 1$. This value of o is rather low, indicating that little information is obtainable from the context of a pixel. This is not surprising considering that the low-order few bits are undoubtedly very noisy. A linear treatment with the assumption of additive Gaussian noise would probably be much more approprate for this kind of data. On the other hand, discarding the lower order bits to give a 4 bit pixel eliminates most of this noise, making the coding scheme perform much better—1.923 bits/pixel, or 48 percent of the unencoded value. We suspect that this may be better than could be achieved using techniques such as linear prediction [8].

Selection of the Escape Probability

We have investigated the use of two algorithms for calculating the escape probability, that is, the probability that a character will occur in a context in which it has not occurred before. The two methods, called A and B, were described above. In practice, we find that there is no clear choice between them. This can be seen in Table II, which compares the best compressions achieved by the two techniques on each of the messages. Method B is slightly better than A on five of the texts and worse on four. Also, there is no apparent relation between the length and type of message and which escape technique fares better. This insensitivity to the escape probability calculation is actually quite satisfying. It illustrates that the coding method is robust, gaining its power from the idea of the escape mechanism rather than the precise details of the algorithm used to implement it. This point is futher reinforced by Roberts [20], who used a very different technique of "blending" Markov models of different orders to achieve excellent results (unfortunately on texts which are not easily comparable with those used here). This insensitivity is particularly fortunate in view of the fact noted earlier that it is hard to see how any particular escape algorithm can be justified theoretically.

Fig. 2 shows graphs of the coding performance versus value of o for both methods, using the text of samples 1 and 3. The general behavior shown there is typical of that for all the examples. In each case method B is relatively less efficient for small values of o but more efficient for large ones. Also,

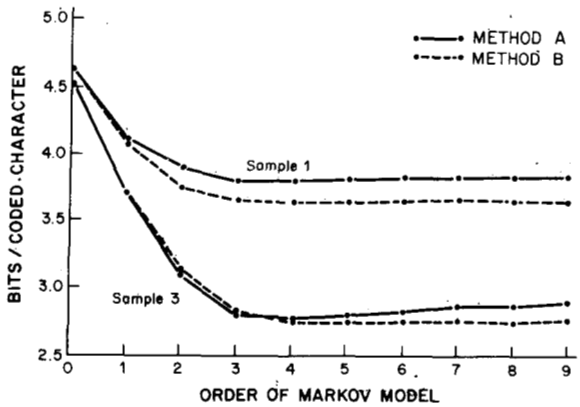

Fig. 2. Coding performance plotted against o, for samples 1 and 3.

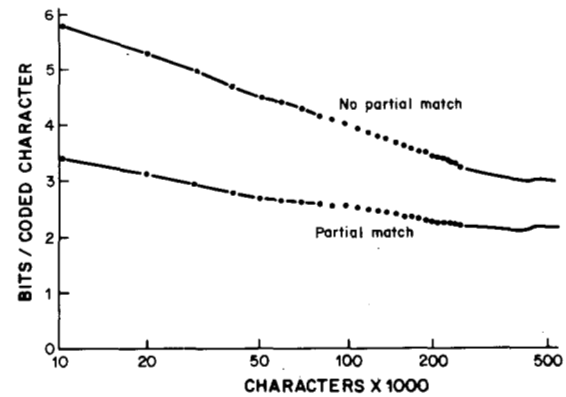

Fig. 3. Cumulative coding performance plotted against time, for sample 2.

method B's efficiency does not deteriorate so quickly past the optimum value of o. This relative lack of sensitivity to o once it is large enough may make Method B preferable in situations where it is hard to estimate the best value for o.

Evaluation of Partial Matching

Recall that the coding scheme uses a "partial match" strategy, whereby it begins forming a model of the desired order at once but uses partial string matching to force predictions out of the nascent model in the early stages. The value of this approach is demonstrated in Fig. 3, which shows how the coding performance varies as time progresses during the long text of sample 2. Time, in terms of number of characters, is plotted horizontally on a logarithmic scale. The vertical axis represents coding performance over the entire initial substring of the message. The lower line shows the performance of the partial string match algorithm, while the same algorithm is used for the upper line but with partial string matching suppressed. In both cases, o was chosen to be 4.

It is partial string matching which allows efficient coding to be achieved early on in the message. For example, the bit rate in the first 10 000 characters is below 3.5 bits/character with partial string matching, whereas without it, it exceeds 5.5 bits/character. Moreover, the improved performance of partial string matching can be seen throughout this rather long piece of text. Eventually, of course, if the message really does have an homogeneous structure, partial string matching will cease to give any advantage. But Fig. 3 indicates that this will take a long time, even for a fairly modest value of o ($o = 4$).

There is an upturn in both lines between 320 000 and 550 000 characters. This is caused by a sudden disruption of the statistics of the text at around character 450 000, which the interested reader will find just over halfway through Chapter 9 of the book [23]. Perhaps this should be taken as a warning that text statistics in real life are not homogeneous and nicely behaved, making it particularly appropriate to use an adaptive encoding method.

IV. Resource Requirements

Let us now consider the resources required to run the coding algorithm. Its most important feature, from the point of view of practical coding, is that the time required for both encoding and decoding grows only linearly with the length of the message. Furthermore, it can be implemented in such a way that it grows only linearly with the order of the model. And impressive data compression has been demonstrated with models of low order—$o = 3$ or 4.

Our current implementation is experimental and inefficient. It is written in the Pascal language on a VAX 11/780 computer. For models of order between $o = 0$ and $o = 4$, encoding time is on the order of 10–50 ms/character, or 20–100 characters/s. Decoding takes a similar time. However, the performance of other implementations of parts of the system have been investigated previously in different contexts. In an Algol implementation, the partial string match search has been found to be possible in 9 ms/character, even for an eighth-order model, on a B6700 computer. We believe that it would be possible to reduce the time taken for partial string matching by the present program by a factor of ten, using better algorithms and hand-coding of critical parts. A tightly coded assembly-language program for arithmetic coding has already achieved 120 μs/character for encoding and 150 μs/character for decoding on a VAX 11/780. Since partial string matching and arithmetic coding between them cover the whole operation of the scheme, a complete data-compression system could operate at approaching 1000 characters/s. Special-purpose architectures using VLSI can be envisaged which could increase the speed to 100 000 characters/s.

The second important resource is the memory space required by both encoder and decoder. As is common in Markov models, this can grow exponentially with the order of the model, and is quite large in practice even for order-5 models of English text. However, notice that the scheme uses no prestored statistics; the required memory is empty initially. There are complicated tradeoffs between space, time, and implementation complexity in partial string matching algorithms [2], [4]. Our experimental implementation stores the Markov model in a tree structure (as must any implementation whose execution time grows at most linearly with o and which occupies a reasonable space).

For each sample of data, the number of nodes in the tree is shown in Table III for various orders of model. All results reported in this paper have been obtained with less than 200 000 nodes. Our experimental implementation in Pascal consumes 128 bits/node, but this can easily be improved. At each node must be stored a character code, a count of the number of times that node has been visited (to allow probabilities to be calculated), and two pointers—one to indicate the next node at the current level and the other to show the subtree for the next level. Allowing 32 bits for the count and each pointer, and 8 bits for the character code, the node consumes 104 bits of storage. For an implementation which accommodates 200 000 nodes (the maximum attained in any of our examples), only 18 bits are required for each pointer. Furthermore, the count could safely be reduced to the same figure or less, on the basis that limiting the counts to even a small maximum value would probably not impair coding efficiency significantly. This would reduce the storage for each node to about 54 bits, so that 1.4 Mbytes would suffice for 200 000 nodes.

TABLE III
SIZE OF DATA STRUCTURE REQUIRED FOR DIFFERENT KINDS OF MESSAGE

data	chars	nodes in tree				
		o=0	o=1	o=2	o=3	o=4	o=9
1. English text	3,614	61	550	1966	4014	6677	22358
2. English text	551,623	94	2417	16802	61546	154077	--
3. English text	44,871	76	1102	5822	17196	35793	198628
4. C source program	3,913	69	625	1733	3236	5062	17571
5. Bibliographic data	20,115	76	1051	4805	11397	19915	80610
6. Numeric data in binary format	102,400	255	14150	48580	--	--	--
7. Binary program	21,505	255	5139	14140	25716	38789	114914
8. Grey-scale picture as 8-bit pixel values	65,536	140	6664	40376	95980	159411	--
9. Grey-scale picture as 4-bit pixel values	65,536	15	212	1600	6634	17912	--

Alternative implementations, such as the compact hash described by Cleary [3], [4], could reduce this to an estimated 28 bits/node and 0.7 Mbytes without sacrificing search time.

Modifications could be made to the coding method which reduce the number of nodes needed. One example is partial model storage. An order-1 model is stored initially. Only when this has been seen to give ambiguous predictions is it augmented to an order-2 model, and then only for the contexts in which ambiguity arises. In general, the order of each node in the model is increased selectively, up to a maximum value of o, whenever more than one prediction is seen to emanate from it. Another possibility is to construct a nondeterministic automaton model of the message string, and store a reduced form as described by Witten [22].

However, we are not overly concerned about the amount of storage that the method consumes. After all, only *unfilled* storage is needed. With the continued improvement in integrated circuit technology, empty store is becoming a cheap resource. The major expense associated with memory is the cost of filling it with information and maintaining and updating that information. But this is done automatically by the coding scheme.

Most coding methods do exact a cost by requiring statistics to be calculated and stored before coding begins. The one described does not. However, many applications will find it worthwhile to prime the encoder and decoder with representative statistics before transmitting a message. This is easily done by sending a representative sample of text before the main transmission begins, and we saw during discussion of Table I that this can be most effective. If the statistics are misleading, then of course some deterioration in coding efficiency is only to be expected. Adaptation will ensure that the initial priming is eventually outweighed by the statistics of the message itself.

APPENDIX

A formal definition is now given of the probabilities estimated for characters using partial string matching. To do this we extend the notation used earlier. Let $c_m(\varphi)$ be the count of the number of times the character φ has occurred in the current context of an order m model; where $0 \leqslant m \leqslant o$ and o is the maximum order of the stored model. In order to gracefully cover the special case when a character has never occurred before in the message (so that $c_m(\varphi) = 0$ for all the models), m is allowed to range from -1 and $c_{-1}(\varphi)$ is defined to be 1 for all φ in the alphabet.

Let the set of characters predicted by the model of order m but not by higher order models by A_m. Then using method A, A_m is the set of characters which have counts greater than 0 in the order m model less all those with counts greater than 0 in higher order models:

$$A_m = \{\varphi: c_m(\varphi) > 0\} - \bigcup_{l=m+1}^{o} A_l.$$

The sets A_m will allow the probability predictions to be improved by neglecting characters predicted by higher order models when calculating the probabilities predicted by the lower order models. For example, if the current context is "# i" and the sequence "i s" has occurred previously in the message but not the sequence "# i s", then "s" is a member of A_1 but not of A_{-1}, A_0, or A_2. Because of the definition above, no character can occur in more than one set A_m. Also because of the definition of c_{-1}, every character in the alphabet will occur in precisely one of A_{-1} through A_0.

Following method A, the probability for a character relative to a model of order m is estimated to be

$$p_m(\varphi) = \frac{c_m(\varphi)}{1 + C_m}, \qquad \varphi \in A_m$$

where C_m is the total count for characters first predicted by a model of order m;

$$C_m = \sum_{\varphi \in A_m} c_m(\varphi).$$

This gives the estimated escape probability of a novel character occurring relative to order m as

$$e_m = 1 - \sum_{\varphi \in A_m} p_m(\varphi) = \frac{1}{1 + C_m}.$$

Finally, the estimated probability for a character using partial string matching is

$$p(\varphi) = p_m(\varphi) \cdot \prod_{l=m+1}^{o} e_l, \qquad \varphi \in A_m.$$

In other words, to compute $p(\varphi)$ start at the highest order o. Reducing the order at each step, take the product of the escape probabilities until the character is positively predicted. Then multiply this product by the probability estimated for the character using the model which first positively predicts it. An example calculation of $p(\varphi)$ using a small alphabet of six characters, "abcdef", is given in Table IV.

TABLE IV
CALCULATION OF PARTIAL STRING MATCH PROBABILITIES USING METHOD A

$c_m(\varphi)$ | φ	a	b	c	d	e	f	C_m	e_m
m								
3	2(2/3)	0	0	0	0	0	2	1/3
2	4	1(1/4)	2(1/2)	0	0	0	3	1/4
1	5	3	2	1(1/2)	0	0	1	1/2
0	5	3	2	1	0	0	0	1
-1	1	1	1	1	1(1/2)	1(1/2)	2	-
$p(\varphi)$	2/3	1/12	1/6	1/24	1/48	1/48		

Values of $p_m(\varphi)$ are given in brackets

ACKNOWLEDGMENT

We would like to thank R. Neal for providing timing data on his fast implementation of arithmetic coding on the VAX 11/780 and an anonymous referee for pointing out the early work on the zero frequency problem by Kant and others.

REFERENCES

[1] L. R. Bahl *et al.*, "Recognition of a continuously read natural corpus," in *Proc. IEEE Int. Conf. Acoust., Speech, Signal Processing*, Tulsa, OK, Apr. 1978, pp. 418–424.

[2] J. G. Cleary, "An associative and impressible computer," Ph.D. dissertation, Univ. Canterbury, Christchurch, New Zealand, 1980.

[3] ——, "Compact hash tables," *IEEE Trans. Comput.*, to be published.

[4] ——, "Representing trees and tries without pointers," Dep. Comput. Sci., Univ. Calgary, Calgary, Alta., Canada, Res. Rep. 82/102/21, Sept. 1982.

[5] J. G. Cleary and I. H. Witten, "Arithmetic, enumerative and adaptive coding," *IEEE Trans. Inform. Theory*, to be published.

[6] T. M. Cover, "Enumerative source encoding," *IEEE Trans. Inform. Theory*, vol. IT-19, pp. 73–77, Jan. 1973.

[7] M. Guazzo, "A general minimum-redundancy source-coding algorithm," *IEEE Trans. Inform. Theory*, vol. IT-26, pp. 15–25, Jan. 1980.

[8] C. W. Harrison, "Experiments with linear prediction in television," *Bell Syst. Tech. J.*, pp. 764–783, July 1952.

[9] R. Hunter and A. H. Robinson, "International digital facsimile coding standard," *Proc. IEEE*, vol. 68, pp. 854–867, July 1980.

[10] C. B. Jones, "An efficient coding system for long source sequences," *IEEE Trans. Inform. Theory*, vol. IT-27, pp. 280–291, May 1981.

[11] B. W. Kernighan and D. M. Ritchie, *The C Programming Language*. Englewood Cliffs, NJ: Prentice-Hall, 1978.

[12] G. G. Langdon and J. Rissanen, "Compression of black-white images with arithmetic coding," *IEEE Trans. Commun.*, vol. COM-29, pp. 858–867, June 1981.

[13] G. G. Langdon, "A note on the Ziv–Lempel model for compressing individual sequences," *IEEE Trans. Inform. Theory*, vol. IT-30, pp. 284–287, Mar. 1983.

[14] M. E. Lesk, "Refer," in *Unix Programmers Manual*, Bell Lab., Murray Hill, NJ, 1979.

[15] R. Pasco, "Source coding algorithms for fast data compression," Ph.D. dissertation, Stanford Univ., Stanford, CA, 1976.

[16] C. S. Pierce, "The probability of induction," in *The World of Mathematics*, vol. 2, J. R. Newman, Ed. New York: Simon and Schuster, 1956, pp. 1341–1354.

[17] J. Raviv, "Decision making in Markov chains applied to the problem of pattern recognition," *IEEE Trans. Inform. Theory*, vol. IT-13, pp. 536–551, Oct. 1967.

[18] J. J. Rissanen, "Arithmetic codings as number representations," *Acta. Polytech. Scandinavica*, vol. Math. 31, pp. 44–51, 1979.

[19] J. J. Rissanen and G. G. Langdon, "Universal modeling and coding," *IEEE Trans. Inform. Theory*, vol. IT-27, pp. 12–23, Jan. 1981.

[20] M. G. Roberts, "Local order estimating Markovian analysis for noiseless source coding and authorship identification," Ph.D. dissertation, Stanford Univ., Stanford, CA, 1982.

[21] H. E. White, "Printed English compression by dictionary encoding," *Proc. IEEE*, vol. 55, pp. 390–396, 1967.

[22] I. H. Witten, "Approximate, non-deterministic modelling of behaviour sequences," *Int. J. General Syst.*, vol. 5, pp. 1–12, Jan. 1979.

[23] I. H. Witten, *Principles of Computer Speech*. London, England: Academic, 1982.

[24] J. Ziv and A. Lempel, "Compression of individual sequences via variable-rate coding," *IEEE Trans. Inform. Theory*, vol. IT-24, pp. 530–536, Sept. 1978.

★

John G. Cleary received the B.Sc. (hons.) and M.Sc. degrees in mathematics and the Ph.D. degree in electrical engineering from the University of Canterbury, Christchurch, New Zealand, in 1971, 1974, and 1980, respectively.

From 1975 to 1981 he worked for a software house, Progeni Systems Ltd., Wellington, New Zealand. In 1981 he was a Lecturer in Computer Science at the Victoria University of Wellington, Wellington, New Zealand. In 1982 he moved to the University of Calgary, Calgary, Alta., Canada, where he is currently an Assistant Professor. His current research interests include adaptive systems and their applications in man–machine interfaces, logic programming and formal specification, and applications of parallel processor systems.

★

Ian H. Witten (M'82) received the B.A. (first class honours) degree in mathematics from Cambridge University, Cambridge, England, in 1969, the M.Sc. degree in computing science from the University of Calgary, Calgary, Alta., Canada, in 1970, and the Ph.D. degree in electrical engineering from Essex University, Colchester, England, in 1976.

A Lecturer and subsequently Senior Lecturer in the Department of Electrical Engineering Science at Essex University from 1970 to 1980, he has recently returned to Calgary to take up a Professorship in the Department of Computer Science. His research interests span the field of man–machine systems and he has specialized in fundamental problems of human and machine learning, in computational phonetics (the science of speech synthesis by computer), in signal processing, and, recently, in document preparation systems and documentation graphics. He is the author and co-author of about 80 publications, including two books, *Communicating with Microcomputers* (New York: Academic, 1980) and *Principles of Computer Speech* (Academic, 1982).