A note on the DMC data compression scheme

T. BELL AND A. MOFFAT*

Department of Computer Science, University of Canterbury, Christchurch, New Zealand

Until recently, the best schemes for text compression have used variable-order Markov models, i.e. each symbol is predicted using some finite number of directly preceding symbols as a context. The recent Dynamic Markov Compression (DMC) scheme models data with Finite State Automata, which are capable of representing more complex contexts than simple Makov models. The DMC scheme builds models by 'cloning' states which are visited often. Because patterns can occur in English which can be recognised by a Finite State Automaton, but not by a variable-order Markov model, the use of FSA models is potentially more powerful.

A class of Finite State Automata, called Finite Context Automata, is defined, which are equivalent to variable-order Markov models. For the initial models proposed, the DMC scheme is shown to have no more power than variable-order Markov models by showing that it generates only Finite Context Automata. This is verified in practice, where experiments show that the compression performance of DMC is comparable to that of existing variable-order Markov model schemes. Consequently, more sophisticated models than Finite Context Automata still need to be explored in order to achieve better text compression.

Received November 1986, revised May 1987

1. INTRODUCTION

Rissanen and Langdon[5] have described an approach to data compression where a model is created of the data to be compressed, and the data are coded with respect to the model. Because efficient methods exist for coding, the main problem has been to find suitable models.

A simple model might assign a fixed probability to each input symbol. A more sophisticated model assigns each symbol a different probability for each possible preceding symbol. This is called a first-order Markov model. In general, Markov models predict a symbol using some finite number of immediately preceding symbols, which are called the Markov context. The larger the Markov context used to predict the next symbols, the less code space is needed to code each symbol. Unfortunately, the use of very large contexts is difficult in practice because a large sample would be required to estimate suitable probabilities accurately, and a large amount of memory would be required to store them.

A popular approach is to use 'variable-order Markov modelling', where for each input symbol the model identifies the longest Markov context that it can estimate probabilities for, and uses that context for coding. Langdon and Rissanen use a combination of zero- and first-order Markov contexts in their 'Double-adaptive File Compression Algorithm'.[4] Cleary and Witten achieved good results with a Markov model which used contexts typically ranging from zero- to fourth-order.[2] Even the Ziv–Lempel coding schemes have been shown to have underlying variable-order Markov models, typically using approximately third-order contexts for prediction.[1]

Although Markov models have been successful, they only approximate to the true behaviour of English text. For example, the fact: 'opening quotations (and parentheses) are almost always followed by closing quotations (and parentheses)' cannot be captured in a Markov model. Also, a verb may have a lot of influence on a noun, regardless of the number of adjectives in between. For example, in 'I stroked the fat cat' and 'I stroked an old black cat', a Markov model would give the greatest weighting to the adjectives 'fat' and 'black' to predict the word 'cat', when it is really predicted by the verb 'stroked'.

Cormack and Horspool describe the Dynamic Markov Compression (DMC) scheme, an adaptive data compression scheme which uses a Finite State Automaton (FSA) as the model.[3] The scheme starts with a simple initial model, and adaptively adds states by 'cloning' states under certain conditions. This approach is potentially more powerful than variable-order Markov modelling because an FSA can represent the type of dependencies just described. The purpose of this paper is to show that, surprisingly, the DMC scheme does not use the full power of an FSA, but is actually a variable-order Markov model.

This is done by defining a subset of FSAs, called Finite Context Automata (FCAs) in Section 2, which are shown to be equivalent to variable-order Markov models. In Section 3 DMC (with its simplest initial model) is shown to be equivalent to variable-order Markov models by showing that the initial model is an FCA, and that cloning states in an FCA always produces another FCA. In Section 4 a class of FSAs is defined which includes all initial models proposed for DMC, and the main theorem is extended to include these initial models. This is done by viewing DMC models as having transitions on symbols rather than bits, where a symbol is a fixed-size string of bits (typically 8-bit bytes).

Definitions

A Finite State Automaton (FSA) is a quadruple (S, A, μ, s), where S is a finite set of states, A is a finite set of symbols (alphabet),

$$\mu: S \times A \rightarrow S$$

is the next state function, and $s \in S$ is the starting state.

Let A^* be the set of all strings, including the empty

* Present address: Department of Computer Science, University of Melbourne, Parkville, Victoria 3052, Australia.

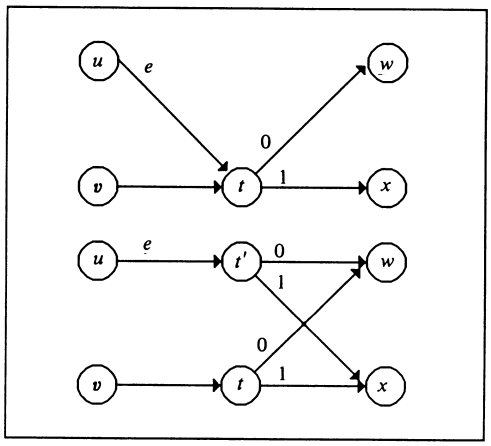

Figure 1. The DMC cloning operation.

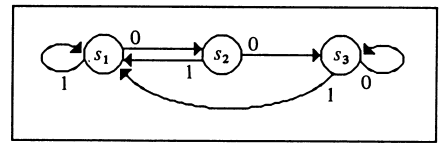

Figure 2. An FSA generated by DMC.

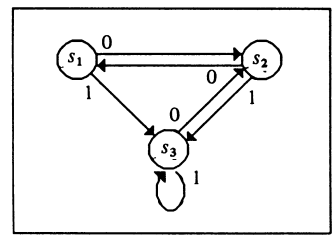

Figure 3. An FSA.

string, with elements drawn from A. The empty string is denoted as Λ. For convenience, μ is extended to A^*:

$$\mu: S \times A^* \to S$$

by the recursive definition:

$$\mu(s, \Lambda) = s,$$

$$\mu(s, p.a) = \mu(\mu(s, p), a) \quad \text{for} \quad p \in A^*, a \in A.$$

The DMC scheme uses an FSA (S, B, μ, s), where $B = \{b_1, b_2, \ldots, b_k\}$. For practical reasons, B is normally the binary alphabet $\{0, 1\}$. Each transition in the FSA has a probability assigned to it based on how frequently it is used. The cloning function takes an FSA and a heavily used transition, $\mu(u, e)$, and creates a new state t', as shown in Fig. 1 for $B = \{0, 1\}$. The new state is created so that statistics for the heavily used transition can be recorded separately, which should, in turn, lead to a better model of the input data and thus a more concise output representation.

Formally, the cloning function, δ, takes the FSA, (S, B, μ, s), a state/symbol pair (u, e), $u \in S, e \in B$, and generates a new FSA, $\delta((S, B, \mu, s), (u, e)) = (S', B, \mu', s))$, with

(1) $\quad S' = S \cup \{t'\}$

(2) $\mu'(u, e) = t',$

$$\mu'(s, a) = \mu(s, a), \quad s \in S, a \in B, s \neq u \quad \text{or} \quad a \neq e,$$

$$\mu'(t', a) = \mu'(t, a), \quad t = \mu(u, e), \quad a \in B.$$

In what follows, the concatenation of strings is extended to the concatenation of sets of strings. If $\{l_1, l_2, l_3 \ldots l_q\}$ is a set of q strings, and m is a string, then

$$\{l_1, l_2, l_3 \ldots l_q\} \cdot m = \{l_1.m, l_2.m, l_3.m \ldots l_q.m\}.$$

If $\{m_1, m_2, m_3 \ldots m_r\}$ is a set of r strings, then

$$\{l_1, l_2, l_3 \ldots l_q\}.\{m_1, m_2, m_3 \ldots m_r\} = \{l_i.m_j: \forall i = 1..q, j = 1..r\}$$

Concatenation with the empty set yields the empty set.

2. FINITE CONTEXT AUTOMATA

Let $F = (S, A, \mu, s)$ be a deterministic FSA. For a machine F there will be a set (possibly empty) of strings $l \in A^*$ that have the special property of forcing F into a particular state, no matter what original state F is in, i.e. $\mu(s_i, l) = \mu(s_j, l)$ for all $s_i, S_j \in S$. Strings outside this set, i.e. non-synchronising strings, are of special interest, and will be defined to be the set $D(F)$:

$$D = D(F) = \{l \in A^*: \mu(s_i, l) \neq \mu(s_j, l), \text{ for some } s_i, s_j \in S\}$$

For example, for the model in Fig. 2, which might have been generated using the DMC algorithm, $D = \{\Lambda, 0\}$, and for the model in Fig. 3, $D = \{\Lambda, 0, 00, 000, \ldots 0^k \ldots\}$.

Define a *Finite Context Automaton* (FCA) to be an FSA, F, where $D(F)$ is finite.

For an FSA, for any $m \notin D$, $\mu(s_i, m)$ is always the same, regardless of s_i, so it can be denoted as $\mu(m)$. For a particular state s, define its context to be the set of all strings which are in this manner guaranteed to result in transitions terminating at s, i.e.

$$\text{context}(s) = \{m: m \notin D \quad \text{and} \quad \mu(m) = s\}.$$

Define

$$c(s) = \text{context}(s) \cap (A.D - D),$$

where, informally, $c(s)$ is a minimal set of suffixes which will result in transitions to state s from anywhere in the machine.

In Fig. 2, $c(s_1) = \{1\}$, $c(s_2) = \{10\}$ and $c(s_3) = \{00\}$, and in Fig. 3, $c(s_1) = \{100, 10000, \ldots 1(00)^k \ldots\}$, $c(s_2) = \{10, 1000, \ldots 10(00)^k \ldots\}$, $c(s_3) = \{1\}$.

From the above definitions, observe that for an FCA, where D is finite, that:

(*a*) If $m \in c(s)$ then $\mu(s, A^*m) = s$, i.e. $A^*m \in \text{context}(s)$.

(*b*) $c(s)$ is finite

(*c*) $\bigcup_i c(s_i) = A.D - D$

(*d*) $c(s_i) \cap c(s_j) = \varnothing$ for $s_i \neq s_j$

Observations (*a*) and (*b*) together imply that the current state is always determined by some finite-size suffix of the input string, and that any information preceding this $c(s_i)$ will not be taken into account in the encoding. In DMC models, each state assigns probabilities to each outgoing transition. Showing that every DMC model is an FCA will be sufficient to demonstrate that the probability of each symbol is chosen based on some finite number of preceding symbols, that is, that DMC is a variable-order Markov model.

Observations (*c*) and (*d*) show that the context function partitions the set A^* into the context sets, and the finite set D.

3. MAIN THEOREM

Before proving the main theorem, the following observations are made about the cloning function.

Observation 1

Transitions on strings in the cloned model which do not end at the state t' end up at the same state as they did before cloning, i.e.

$$\text{For } s \in S, \quad m \in B^*, \quad \mu'(s,m) \neq t' \Rightarrow \mu'(s,m) = \mu(s,m).$$

This happens because on each transition the μ' function is only different from the μ function when the transition is to t'. Because the new transitions out of t and t' are the same as the old transitions out of t, strings which pass through those states in the cloned model leave from them to the same state as they did in the original model.

Observation 2

For $s \in S$, $m \in B^*$, if $\mu(s,m) \neq \mu'(s,m)$ then $\mu(s,m) = t$ and $\mu'(s,m) = t'$. This follows from the converse of Observation 1, given that $t' \notin S$.

Lemma 1

Let l be a string of k symbols, and p be the first $k-1$ of these, i.e. $l = p.a$, $p \in B^*$, $a \in B$. If $\mu'(s_i, l) = t$, and $\mu'(s_j, l) = t'$, then $\mu(s_i, p) \neq \mu(s_j, p)$.

Proof

Suppose $\mu(s_i,p) = \mu(s_j,p)$. Since the machine is deterministic and $\mu'(s_i,l) \neq \mu'(s_j,l)$, it must be that $\mu'(s_i,p) \neq \mu'(s_j,p)$. By Observation 2, the only target which is different after cloning is t, which afterwards is t or t'. This implies that

$$\mu(s_i,p) = \mu(s_j,p) = t,$$

and

$$\mu'(s_i,p) = t,\ \mu'(s_j,p) = t' \text{ (or } \mu'(s_i,p) = t',\ \mu'(s_j,p) = t),$$

which implies:

$$t' = \mu'(s_j, p.a) = \mu'(\mu'(s_j,p), a) = \mu'(t', a) = \mu'(t, a)$$
$$= \mu'(\mu'(s_i,p), a) = \mu'(s_i, p.a) = t,$$

which is a contradiction. A similar contradiction occurs if $\mu'(s_i,p) = t'$ and $\mu'(s_j,p) = t$, so the lemma follows.

Main theorem

Every model generated by DMC when started on the initial 'one-state Markov model', $F = (S, B, \mu, s_1)$, is an FCA, where $S = \{s_1\}$, $B = \{b_1, b_2 \ldots b_k\}$, $\mu(s_1, b_i) = s_1$.

Proof

The proof is by induction. The initial model, F, has one state only, s_1, and every string leads to that state. This means that $D = \varnothing$, which is finite, so F is an FCA. The rest of the proof shows that cloning an FCA produces an FCA, by showing that if D is finite then D' is finite.

$$D' = D((S', B, \mu', s_1))$$
$$= \{l: \mu(s_i,l) \neq \mu'(s_j,l),\ s_i, s_j \in S'\}$$

By Observation 2, the only strings with new targets are those with t or t' as their targets, and the empty string. Hence,

$$D' = \{l: \mu(s_i,l) \neq \mu(s_j,l),\ s_i, s_j \in S\} \cup \{l: \mu'(s_i,l) = t,\ \mu'(s_j,l) = t',\ s_i, s_j \in S'\} \cup \{\Lambda\}$$

The first term is the set D, and the middle term can be altered using Lemma 1, giving

$$D' \subseteq D \cup \{l: 1 = p.a,\ \mu(s_i,p) \neq \mu(s_j,p),\ s_i, s_j \in S,\ a \in B\} \cup \{\Lambda\}$$
$$\subseteq D \cup \cup_{a \in B}\{p: \mu(s_i,p) \neq \mu(s_j,p),\ s_i, s_j \in S\}.a \cup \{\Lambda\}$$
$$= D \cup D.B \cup \{\Lambda\}$$

which is finite.

This then completes the proof.

4. OTHER INITIAL MODELS

Although the 'one-state Markov model' (Fig. 4*a*) is a suitable initial model, Cormack and Horspool achieved slightly better compression by using more sophisticated initial models which correspond to bytes or words, rather than bits. In order to do this, and still retain the simple alphabet of $B = \{0, 1\}$, these initial models contain cycles of h bits, typically with $h = 8$. One of the structures suggested is a 'tree', which is shown in Fig. 4*b*, for $h = 3$. Also suggested is a 'braid' structure, which is a generalisation of a tree, with h levels (typically 8), and 2^h nodes at each level (256). A given h-bit sequence follows transitions from any top-level node down the braid, and back to a *unique* top-level node.

We are most interested in behaviour at the h-bit symbol level, where h is the length of cycles in the initial model. With this object, it is convenient to redefine the transition and cloning functions to allow for transitions drawn from the alphabet $A = B^h$, bit strings of length h. Fig. 5 shows an initial model, F_1, and its corresponding 3-bit symbol-level equivalent. Viewing DMC models at the symbol level is possible because cloning only creates cycles with lengths which are multiples of h, so only

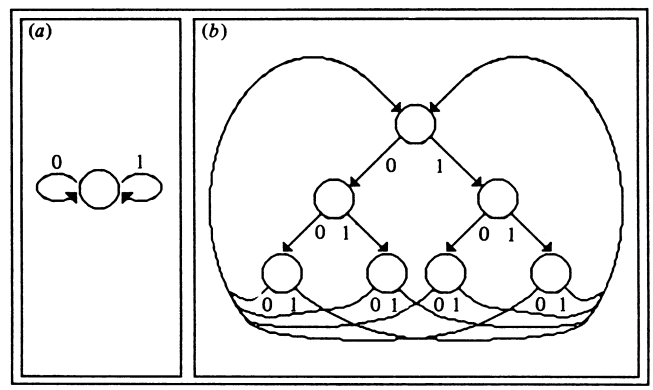

Figure 4. Initial models proposed for DMC: (*a*) one-state Markov model; (*b*) binary tree.

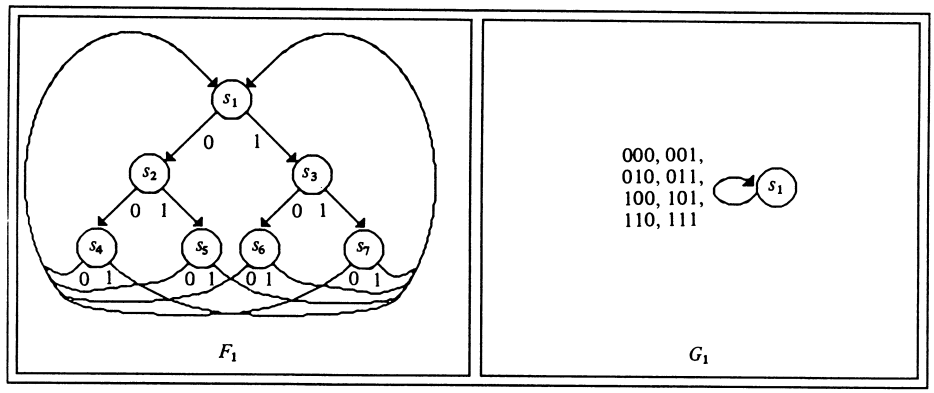

Figure 5. An initial binary tree of depth 3 (F_1), and its symbol-level equivalent (G_1).

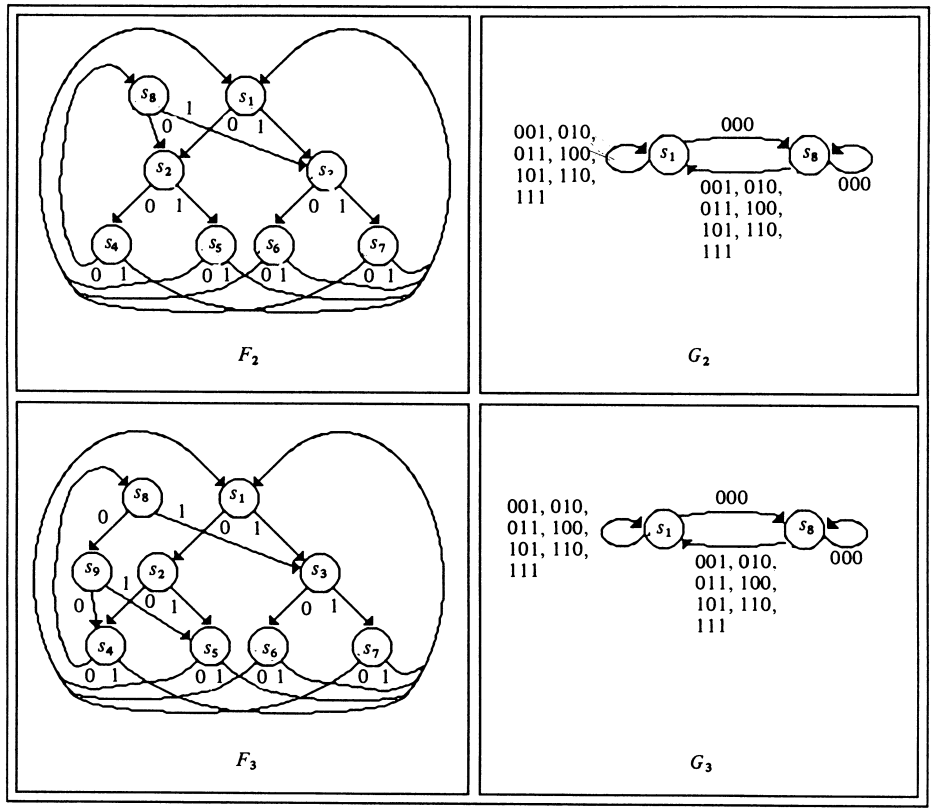

Figure 6. The effect of cloning on a symbol-level equivalent. G_2 and G_3 are the equivalents of F_2 and F_3 respectively. $F_2 = \delta(F_1, (s_4, 0))$, $F_3 = \delta(F_2, (s_8, 0))$, $G_2 = \delta_h(G_1, (s_1, 000))$, $G_3 = G_2$.

nodes cloned from the starting state, s_1, are visited after each h bits of input. This can be verified by considering the effect of cloning a state which is part of a cycle (see Fig. 1). Fig. 6 shows how cloning s_1 is reflected in the symbol-level equivalent, G_2, and how cloning other states has no effect on the equivalent (G_3). This new view ignores dependencies between bits, but retains the model structure as far as h-bit symbols are concerned.

The set of nodes cloned from s_1 is the set of all nodes visited from s_1 after a multiple of h bits:

$$S_h = \{s: \exists m \in A^*, \mu(s_1, m) = s\}$$

The symbol-level transition function is

$$\mu_h: S_h \times A^* \to S_h,$$

$$\mu_h(s, l) = \mu(s, l),\ s \in S_h,\ l \in A^*.$$

A bit-level cloning only results in a change at the symbol level when the cloning is on a transition which is the last bit of a symbol. Thus the bit-level cloning $\delta(u, e)$ transforms the symbol-level model, F, to F', where

$$F' = \delta_h(F, U) \quad \text{if} \quad \mu(u, e) \in S_h, \quad \text{with } U = \{(s, a): s \in S_h,$$

$$a = a_1 a_2 \ldots a_h, \mu(s, a_1 a_2 \ldots a_{h-1}) = u, a_h = e\};$$

$F' = F$ otherwise.

The new cloning function, $\delta_h(F, U)$, is very similar to δ. $\delta_h((S_h, A, \mu_h, s_1), U) = (S'_h, A, \mu'_h, s_1)$, where

(1) $S'_h = S_h \cup \{t'\}$

(2) $\mu'_h(u, e) = t'\ \forall (u, e) \in U,$

$\mu'_h(s, a) = \mu_h(s, a),\ s \in S_h,\ (s, a) \notin U,$

$\mu'_h(t', a) = \mu'_h(t, a),\ a \in A.$

The two initial models suggested for DMC translate into two types of model at the symbol level, with the new

alphabet $A = B^h$. The 'tree' model corresponds to the zero-order Markov model:

$$F_0 = (S = \{s_1\}, A, \mu_h, s_1),$$

with $$\mu_h(s_1, a) = s_1, \quad \forall a \in A.$$

The 'braid' model corresponds to a first-order Markov model, with 2^h nodes, each node being labelled with one of the 2^h symbols in the alphabet:

$$F_1 = (S = \{s_a : a \in A\}, A, \mu_h, s_1),$$

with $$\mu_h(s, a) = s_a, \quad \forall a \in A, \quad s \in S,$$

and s_1 is any $s_a \in S$.

From this, $D(F_0) = \varnothing$, and $D(F_1) = \{\Lambda\}$, both of which are finite, so all the initial models are FCAs at the symbol level. Of course, initial models which are not FCAS could be supplied to DMC, but it is difficult to find such a model which offers any advantage over the FCA models proposed.

Because the symbol-level view of the model has a cloning function isomorphic to those used in the proofs in Section 3, the results of that section hold for the symbol-level model. In particular, all models generated by DMC are FCAs at the symbol level, when started with any of the suggested initial models. For example, with an 8-bit initial model, every byte (character) is predicted using some finite number of preceding bytes. The probabilities used for prediction will be affected by interactions within the bit patterns of the symbols, but the overall probability used to encode any symbol is selected entirely by a Markov context.

5. CONCLUSION

It has been shown that DMC uses an underlying variable-order Markov model. This has been done by viewing the model at the symbol level rather than the bit level, where the size of a symbol is implied by the initial model. In practice, DMC achieves compression comparable to Cleary and Witten's Partial Match scheme (which also uses a variable-order Markov model); DMC appears to be a little worse for text files, but significantly better for non-homogeneous binary files. Although the DMC scheme models text with an FSA, the FSA is restricted to be an FCA, and the use of unrestricted FSA – and more sophisticated – models of text for data compression still needs to be explored.

Acknowledgements

The authors would like to thank John Cleary for suggesting this line of research, Ian Witten for helpful suggestions and comments on the work, and Nigel Horspool for supplying the DMC program and his helpful comments.

REFERENCES

1. T. C. Bell, A unifying theory and improvements for existing approaches to text compression. Ph.D. thesis, Department of Computer Science, University of Canterbury (1987).
2. J. G. Cleary and I. H. Witten, Data compression using adaptive coding and partial string matching. *IEEE Trans. Comm.* **COM-32** (4), 396–402 (1984).
3. G. V. Cormack and R. N. Horspool, Data compression using dynamic Markov modelling. *The Computer Journal*, **30** (6), 541–550.
4. G. Langdon and J. Rissanen, A double-adaptive file compression algorithm. *IEEE Trans. Comm.* **COM-31** (11), 1253–1255 (1983).
5. J. Rissanen and G. Langdon, Universal modeling and coding. *IEEE Trans. Inf. Theory*, **IT-27**, 12–23 (1981).