ACM Trans. on Information Systems **8** (1990) 27–49.

Compression, Information Theory and Grammars: A Unified Approach

Abraham Bookstein and **Shmuel T. Klein**

Center for Information and Language Studies
University of Chicago, 1100 East 57-th Street,
Chicago, Illinois 60637

January 1990

Abstract:

Text compression is of considerable theoretical and practical interest. It is, for example, becoming increasingly important for satisfying the requirements of fitting a large database onto a single CD-ROM. Many of the compression techniques discussed in the literature are model based. We here propose the notion of a formal grammar as a flexible model of text generation that encompasses most of the models offered before as well as, in principle, extending the possibility of compression to a much more general class of languages. Assuming a general model of text generation, a derivation is given of the well known Shannon entropy formula, making possible a theory of information based upon text representation rather than on communication. The ideas are shown to apply to a number of commonly used text models. Finally, we focus on a Markov model of text generation, suggest an information theoretic measure of similarity between two probability distributions, and develop a clustering algorithm based on this measure. This algorithm allows us to cluster Markov states, and thereby base our compression algorithm on a smaller number of probability distributions than would otherwise have been required. A number of theoretical consequences of this approach to compression are explored, and a detailed example is given.

1. Introduction

Compression is of interest for two reasons: Most immediately, it is of great practical importance, both for the storage and transmission of information. This is likely to continue to be the case for some time into the future. While the significance of compression for data transmission is generally accepted, it is often noted that storage devices are becoming less expensive, and that this limits the need for data compression to save on storage requirements. Less often noted, however, is that our ambition to store information is similarly growing quickly. Being able to store a large database on a single

CD-ROM rather than two has significant implications for its cost and convenience of use [13]. For overviews of compression techniques and theory see [15] or [26].

The possibility of compressing text also has important theoretical implications. Much of the theory of data compression relies heavily on Shannon's theory of information [2], [24], a body of mathematics intended for analyzing the encoding and transmission of messages across a possibly noisy channel. In his formulation, Shannon offered a set of axioms describing the properties desired of a measure of uncertainty within a communications context, and deduced from these axioms a function, H, that he called *Entropy*: if we have a set of messages $M = \{m_i\}$, and each m_i has an associated probability, P_i, then H is defined by the well known formula

$$H = -\sum P_i \log P_i.$$

H quantifies the uncertainty regarding which message will be selected for transmission; the reduction in uncertainty then defines the information content of a transmission. Of particular importance for us, it is subsequently shown that H constitutes a lower bound on our ability to compress data (see, e.g., [8] for a very readable derivation of this result): given a set of items and probabilities, $\{m_i, P_i\}$, it is not possible to usefully encode the items $\{m_i\}$ with a binary code so that the lengths of the codewords, l_i, satisfy $\sum P_i l_i < H$. H thus defines a theoretical limit in the compressibility of a set of messages, given a probability model describing message generation.

We believe that the full conceptual implications of this result have been largely overlooked. The possibility of compressing data provides a basis for a completely new derivation of the entropy formula: given a probabilistic message generator, the uncertainty of (or information contained in) its message set can be *defined* as the smallest amount of storage, on the average, needed to store the encoded output of the machine. We will show that the Shannon measure follows directly from this definition. Thus entropy is directly related to compression, rather than primarily a communication-based concept with incidental implications for compression.

A number of authors have commented on the importance of correct source modeling for good compression [22], [20]. A mechanism for representing source models that we find appealing because of its simplicity and flexibility is that of a grammar, [10], especially for describing text generation. In Section 2, we introduce the notion of a grammar more formally, and show how it can be used to derive Shannon's entropy formula: H then measures the "information content" of the messages generated by

the grammar. Several special cases, including the most popular ones appearing in the literature, are then described from this vantage point.

It is not realistic, however, to contemplate using a grammar defining a natural language as the basis for a practical compression procedure. We therefore present in Sections 4 and 5 a compression method based on a model we believe is a good trade-off between a simple, but not very accurate, model like independent character generation, and a much more sophisticated model like a general grammar, which is impossible to implement. In Section 4, we assume that natural text is generated by a first-order Markov process with anomalies: certain strings, relatively few in number, occur at rates substantially greater than expected on the basis of the Markov assumption. We first identify these strings and replace each of them by a single new symbol added to our alphabet. The new alphabet is then encoded as a first-order Markov process; for efficiency, the number of states is reduced by a clustering mechanism, as described in detail in Section 5. The clustering is based upon a similarity measure inspired by information theoretic arguments. A detailed example of the clustering algorithm is presented in Section 6.

2. Grammars

Our discussion of compression is guided by the notion of a grammar. A grammar is a systematic description of how a language is created. It permits us to define a set with an infinite number of items (the sentences of the language) in a finite number of bytes (the grammar). Knowledge of the grammar supplies a great deal of information about the sentences, and thus reduces the information contained in the sentences themselves. This can be made more precise: we are considering a machine that generates text, where the output of the machine is describable as a tuple $(V_N, V_T, \mathcal{P}, S)$: V_N is a set of variables, V_T a set of terminal symbols, $\mathcal{P}$ a set of productions and $S \in V_N$ a start symbol [10]. We adopt this customary definition, except that we also associate a probability with each production rule. One consequence of the introduction of probabilities into grammars is that it allows us in principle to assign a probability to each message. These message probabilities are the basis of all the compression methods and theory discussed below.

We now show that it is possible, given the message probabilities, to derive the Shannon entropy formula on the basis of compression considerations alone. To do this in the fullest generality, we must first decide what it is that our source is generating. Beginning with the notion of a grammar suggests that the fundamental conceptual unit

be the whole message, not the individual characters. We may well compress a message by a sequence of character oriented steps. But ultimately the message is conceived of as a unit, and after compression we have a bit string representing the full message*.

We begin, then, with the notion of a probabilistic message source, and consider the set of all possible messages $\{M_i\}$ that can be generated by this source; associated with each message M_i is the probability, P_i, that M_i will be generated. Though the computation may be complex, it is in principle possible to derive these probabilities given a grammar for the source. If a limit is imposed on the size of a message, the message set can be considered finite. Also associated with M_i is its compressed representation, a bit string of length ℓ_i. We further assume that none of the bit strings is a prefix of any other—this may require adjoining a unique "end of message" character at the end of every message, as is usually done for arithmetic coding [21], [28]. The average compressed message length is $\sum P_i \ell_i$. The entropy H_G of the source, described by the grammar G, can now be defined as the minimum value this length can take over all possible compression procedures. We can further define the entropy per character as $H_G/\bar{n}$, for $\bar{n}$ the expected length in characters of the message set, provided the latter is defined. We speculate that the value $H_G/\bar{n}$ will be smaller than the corresponding values computed from distributions assuming independent character occurrence. That is, our knowledge of the source permits us to compute the true probability, P_i, of M_i occurring, and hence H_G and $H_G/\bar{n}$. But, given a set of textual messages, we can statistically analyze the occurrences of characters to estimate the global probability of occurrence of each character; denote the probability of occurrence of the i-th character by p_i. Then the entropy per character under the independence model, H_I, is given by $H_I = -\sum p_i \log p_i$; we expect $H_G/\bar{n} \leq H_I$. The difference, $H_I - H_G/\bar{n}$, measures the information per character captured by our knowledge that the language was generated by a grammar. It is a measure of the information content of the grammar.

To evaluate the entropy we first note that the lengths must obey the McMillan inequality: $\sum_i 2^{-\ell_i} \leq 1$ (see, for example, [4, Theorem 4.1]). This is a general property of binary trees (any set of bit strings satisfying the prefix property in effect defines a binary tree). Given any set of codewords, we can create another set of average

---

* This does not foreclose the possibility of generating a number of messages in sequence. However, when we do so, we think of the messages as being generated independently, and the encoding and decoding processes starting over again each time. (This contrasts with the notion of a *code extension*, in which, for encoding purposes, a fixed number of successive independently generated messages is treated as a single message from a correspondingly large message set [8].)

length no longer than the original that also represents our messages, observes the prefix property, and satisfies the McMillan *equality*; that is, the new set corresponds to a *complete* binary tree ([14, Exercise 2.3.4.5–3]). Since we are searching for optimal compression, we can assume the equality is satisfied.

Using Lagrangian techniques** and ignoring integer constraints, one can show that the expected size of a compressed message, $\sum_i P_i \ell_i$, is minimized when $\ell_i = -\log P_i$, where the optimization takes place subject to the McMillan equality; logarithms throughout this paper are to base 2. This immediately shows that $H = -\sum_i P_i \log P_i$ is a lower bound on the average size of encodings of messages from the source. H can be shown to be in fact a *greatest* lower bound (using either code extensions or arithmetic encoding). Other familiar properties of H follow immediately from our derivation. For example, if $\{P_i\}$ and $\{Q_i\}$ are probabilities, then

$$-\sum_i P_i \log \frac{Q_i}{P_i} \geq 0; \tag{1}$$

otherwise, by setting ℓ_i (treated as a continuous variable) equal to $-\log Q_i$, we would have $\sum P_i \ell_i < -\sum_i P_i \log P_i$; but since the set of values, $\{\ell_i\}$, constructed in this way satisfy the McMillan constraint (recall $\sum Q_i = 1$), this contradicts our optimization argument. (A more direct proof of (1) is possible based on Lagrangian methods: it is easy to show that the values Q_i for which $-\sum_i P_i \log \frac{Q_i}{P_i}$ is minimized, subject to $\sum_i Q_i = 1$, is $Q_i = P_i$; for these values of Q_i, the sum in (1) is zero.)

Huffman [11] described an optimal algorithm for compressing data of a given source. The argument that this algorithm provides an optimal code also does not depend on prior information theory based arguments. We see then that considering compression oriented concepts as primary, the formula and properties of H follow independently of the context of communication, and can be used as an alternative development of information theory. An advantage of this approach, besides providing an independent and immediately graspable argument supporting the Shannon formula, is that it brings the theory closer to the heart of theoretical computer science: parallel to the definitions of the time and space complexity of a problem $\mathcal{P}$ in terms of performance measures of optimal algorithms solving $\mathcal{P}$, we define the information content of an information generator in terms of a performance measure of an optimal storage algorithm.

** Lagrangian techniques extend the basic method of differential calculus for finding an unconstrained extremum of a differentiable function $f(x)$ of a vector variable $x = (x_1, x_2, \ldots)$. If the variables x_i are constrained to satisfy, say, a single constraint, $g(x) = 0$, then the maxima and minima of f must satisfy $\frac{\partial L}{\partial x_i} = 0$, where $L = f(x) - \lambda g(x)$ and λ (the Lagrange multiplier) is a constant determined by the constraint (see [7]).

We now consider special cases of grammars as models of text generators.

2.1 Independent character generation

Most compression applications are implicitely based on the assumption that characters are generated independently. This can be represented in terms of a grammar as follows: given an alphabet A of m characters $\{c_1, \ldots, c_m\}$, we have the m productions $\{S \rightarrow c_i S \ \ (P_i)\}$, for S the starting and only non-terminal symbol, $V_T = A$, and P_i referring to the probability of the character c_i occurring. We will arbitrarily stop the process after n characters have been generated, though simple elaborations of this model will generate sentences that have a given expected length without such an external stopping procedure. For example, we could include a special stop-character with a specified probability of occurrence.

2.2 Simple Markov model

A natural generalization of the model of independent character generation is that of a (first or higher order) Markov process [12], [17]. A first-order Markov process is a probabilistic process in which the probability of occurrence of an event is determined only by the immediately preceding event. This model is more flexible than the model of independent character generation, since probabilities are influenced by history; however, the memory of a first-order Markov process is very limited: $\Pr\{x_i \mid x_{i-1}, x_{i-2}, \ldots\} = \Pr\{x_i \mid x_{i-1}\}$, for x_i the state of the system at time i. Higher order Markov processes are immediate generalizations of the first-order process.

A first-order Markov model for an m character alphabet can be represented by a grammar with $V_N = \{S, S_1, \ldots, S_m\}$, $V_T = A$, and having the productions

$$\begin{cases} S & \rightarrow c_i S_i \quad (P_i) \\ S_i & \rightarrow c_k S_k \quad (P_{ik}). \end{cases}$$

For simplicity, we assume this process stops after generating n characters. The generalization to higher order processes, in which the probability of a character depends on a fixed number of preceding characters, is immediate; it is a special case of the model described in the next section.

2.3 Variable length Markov model

We can also represent processes in which the probability of a character depends on a variable number of preceding characters [17]. Such a process can be represented by

a grammar that includes production rules of the following form:

$$\begin{cases} S \rightarrow c_i S_i & (P_i) \\ S_{i_1 i_2 \ldots i_{r-1}} \rightarrow c_{i_r} S_{i_m i_{m+1} \ldots i_{r-1} i_r} & (P_{i_1 i_2 \ldots i_{r-1}\,;\, i_m \ldots i_r}) \end{cases}$$

for some $r \geq 2$ and $1 \leq m \leq r$; that is, we have just scanned the string $c_{i_1} c_{i_2} \cdots c_{i_{r-1}}$, and the probability is $P_{i_1 i_2 \ldots i_{r-1}\,;\, i_m \ldots i_r}$ that c_{i_r} will be generated and a state entered that is defined by the last $r - m + 1$ characters scanned. A k-th order Markov process, with $k \geq 1$, is a special case of this model.

2.4 General grammar

The grammars defined above describe languages that permit the type of sequential encoding/decoding of text that is customary in data compression: one can encode text by scanning characters sequentially and allowing the sequence of characters scanned to define the state of the encoder; this state then determines the probability of occurrence, and thus the codeword, for the next character. But the theoretical strength of grammars is that they in principle permit modelling sources which produce complexly structured text. The following simple example is included to indicate the possibilities inherent in the grammar model; the language it produces cannot be analyzed fully by the types of statistical approaches generally used for compression.

Consider, then, the language: $\{ab, aabb, aaabbb, \ldots\}$, i.e., the language whose alphabet is $\{a, b\}$ and whose sentences are n a's followed by n b's. We can represent this language by a grammar with the following production rules:

$$\begin{cases} S \rightarrow a\,b & (p) \\ S \rightarrow a\,S\,b & (1-p) \end{cases}$$

Thus the number of a's and b's is a random variable, denoted by N. For this simple case, we can very easily compute the probability of each sentence: the sentence made up of n a's followed by n b's has probability $\Pr(N = n) = (1-p)^{n-1}p$ of occurring. Also, since in the final sentences there are as many a's as b's, the "global probability" of each character is 1/2. For the grammar

$$H = -\sum_{n=1}^{\infty} p(1-p)^{n-1} \log(p(1-p)^{n-1}) = \tfrac{1}{p}[-p \log p - (1-p)\log(1-p)].$$

Since for this distribution, $E(N) = \sum_{n=1}^{\infty}(2n)p(1-p)^{n-1} = \frac{2}{p}$, the average entropy for a character is $\frac{1}{2}[-p \log p - (1-p)\log(1-p)] \leq \frac{1}{2}\log 2 = 0.5$. If we had used the

customary independence assumption, with the probability set to the global probability of 1/2, we would have concluded $H = 1$. Thus the grammar resolves at least half of our uncertainty regarding which character will occur next.

Generally, messages are compressed incrementally. The encoder receives one character at a time and either on the basis of a preassigned set of probabilities or adaptively, using limited memory (Lempel & Ziv [29]), adds to the encoded string. Such an analysis is not possible here: once the first b is encountered, the rest of the text is known; but keeping track of how many b's will be needed requires unlimited memory. The current model suggests that radically different approaches to compression may be possible.

3. Notation and Conventions

It will be useful at this point to define notations that will be heavily used below. P's, appropriately subscripted, will denote probabilities. Given a string $s_1 s_2 \cdots s_n$, s_i denotes the i-th character of the string and P_{s_i} its probability of occurrence; $P_{s_i s_j}$ denotes the probability of the character s_j occurring next, given that s_i has just been generated. In this notation, the indices refer to positions in a string. The s_i's are taken from an alphabet A. Sometimes it will be useful to refer to the i-th character of the alphabet, $c_i \in A$, or of some other set of characters. We will use the notation P_i as the unconditional probability that c_i occurs, and P_{ij} the probability that c_j occurs next, given we have just scanned c_i. This notation is extended to denote the probability that any character in $\mathcal{C}$, a set (or *cluster*) of characters, occurs by $P_{\mathcal{C}}$; $P_{\mathcal{C}i}$ will denote the probability of c_i given some $c_j \in \mathcal{C}$ has occurred, as defined below. The $\mathcal{C}$ in our notation reflects that below the sets will be generated by a clustering algorithm.

Below, we shall need an estimate for the length of the codeword of a message with probability P of occurrence; we shall use $-\log P$ for this purpose. This is an idealized length since it represents a lower bound on the size of the codeword. However, this ideal is obtained, or approached, for many codes. For example, this formula is exact for the Huffman code of a dyadic probability distribution, that is, a distribution where each probability is an integral power of 2^{-1} [19]. For example, the probability distribution $(2^{-2}, 2^{-2}, 2^{-2}, 2^{-3}, 2^{-4}, 2^{-4})$ is dyadic. For non-dyadic distributions, there are many ways of justifying this approximation, for example considering code extensions, Shannon-Fano coding [8], or arithmetic coding [21].

We will use Huffman coding (as in [17] or [18]) in our discussion below, though our ideas apply to arithmetic coding (as in [3], [20]) as well. The reason we are emphasizing

Huffman coding rather than arithmetic coding or Ziv & Lempel [29] coding, even though these codes might yield better compression [28], is that Huffman codes always encode a given element in the same way, a desirable property in certain applications (see [13]). Further, in practice, Huffman codes approximate the idealized limit quite well (see the example in Section 6), so the theory should be adequate for these codes.

4. Hybrid Model

Most desirable for compression would be a full probabilistic grammar correctly describing the text being encoded. Lacking this, we must rely on statistical models that capture essential aspects of the text. The simple independence model is clearly inadequate. The first-order Markov model is an appealing substitute. It is a simple generalization of the independence model, and yet captures some of the statistical dependency that is inherent among the components constituting text. However, text is made up of segments with rather long, strong dependencies that a first-order Markov model is incapable of representing. Using higher order or variable length models increases the complexity of the analysis.

We propose here a compromise approach. We recognize the need to encode variable length strings, but carry out this encoding in two stages. First, we identify a small number of strings that occur frequently in our text, and represent them by single symbols not already in our alphabet; we then encode the alphabet enhanced by these symbols as a simple Markov process. Unfortunately, by increasing the size of the alphabet, this strategy also increases, perhaps substantially, the space requirements of auxiliary tables. In Section 5 we will introduce a clustering method that allows us to accommodate a very large alphabet while limiting the size of auxiliary tables.

4.1 Implementation considerations

As mentioned in the introduction to the article, we are assuming that text is generated by a Markov Process. To implement a k-th order Markov model, a distinct table of probabilities of character occurrence must be defined for each string of k characters. After such a string is scanned, the Huffman code for the ensuing character is determined by the probability table of the scanned string. Two problems immediately arise:

(a) One expense of a code is the storage requirement for auxiliary tables, and this varies with the generation model. If we have m characters, we need store only m variable length codewords for the simple independence model; this increases to m^2

codewords for the first-order Markov model and to m^{k+1} for the k-th order Markov model. If m is 100, a million table entries are needed for a second order model, and the code tables themselves become a substantial consumer of space resources. Thus, in practice, one would rarely use higher orders than one. But then:

(b) a first-order Markov model, while it may improve upon the assumption of independent character generation, and perhaps may even be quite adequate in general, fails most conspicuously because of the frequent occurrence of certain strings, especially common trigrams and words. These often occur substantially more frequently than expected from the Markov assumption.

In other words, in creating a compression program, we must resolve two conflicting demands: the order of the Markov process chosen to describe the character generation of the given text should on the one hand be made as low as possible to reduce the space complexity, and on the other hand as high as possible to get a model which is closer to reality. Our two stage procedure offers a trade-off for the above demands.

Certain strings occur more frequently than expected from the Markov assumption. We first extend our alphabet to include these strings. If A is our current alphabet, we procede as follows: if the string $s_1 s_2 \cdots s_n$, for $s_k \in A$, occurs substantially more often than expected on the basis of the Markov model, recode it as a single symbol, S_i, and treat $A \cup \{S_i\}$ as the alphabet to be encoded. Since, in practice, n will be limited in size, this process will terminate if continued iteratively. Only strings causing the largest discrepancies will be transformed in this manner. In the second stage, we apply a clustering mechanism to the expanded alphabet. The difficulties of identifying the strings to replace by single symbols in our first stage, and then of resolving ambiguities inherent in reducing actual text to a sequence of symbols from this new alphabet, have been discussed extensively ([9], [27], [23]). We shall only comment on an aspect of this problem that is illuminated by the information theoretic approach we are taking in this paper.

4.2 Measure of worth

We must identify the strings that are to be replaced by single symbols. Processing all n-grams in order to identify the optimal set is too costly. Fraenkel, Mor & Perl [6] show that even if we restrict the potential n-grams to prefixes and suffixes of the words in the text, the problem of finding an optimal set is NP-complete. One therefore typically uses a heuristic that is reasonably effective. We anticipate that bigrams, trigrams,

and words would be especially practical and useful, so we recommend restricting our n-grams to these.

We next need a measure of worth, w, for each candidate string; w is used to choose which strings to translate. A number of candidate measures are possible.

(a) The most naive approach is to tabulate the number of occurrences of each string (w_0 = frequency of string), and use the most frequent.

(b) But translating a long string to a single codeword may yield a greater savings than translating a shorter, though more frequently occurring string. Therefore a more sensitive, but still easily computable measure, is most commonly used ([6], [23]): $w_1 = (\ell - 1)f$ for a string that is ℓ characters in length and which occurs f times.

The measure w_1 can be justified on two grounds. If we think of the compression process as being implemented in stages, then we first compress a number of strings into one byte codewords. All resulting symbols are then merged with the initial alphabet and the resulting alphabet is finally Huffman encoded. If we procede in this manner, we would like the first stage compression to be as effective as possible; w_1 ranks the strings according to the savings accrued by replacing each string by a single byte. A second motivation is that those strings exhibiting the greatest savings in stage one are likely to be the same as those whose probability of occurrence most exceeds the expected value as predicted by a Markov model.

(c) The last measure of worth, w_2, follows naturally from our discussion of the encoding of a Markov process. We noted that frequency alone is inadequate as a criterion for substitution since the compression effectiveness of reducing a string to a single symbol is affected by its length as well. But also, a string may occur often simply because its components are expected to occur frequently. If the string occurs frequently only because its components do, no earnings occur from reducing the string to a single symbol. Consider, for example, the string $S = s_1 s_2 \cdots s_n$. If this has been generated by the underlying first-order Markov process, the probability with which this string occurs is given by $P_{s_1} P_{s_1 s_2} \cdots P_{s_{n-1} s_n}$. The length of the Huffman encoding of this string as a single unit will be approximately $-\log(P_{s_1} \cdots P_{s_{n-1} s_n})$. If the text is N characters in length and P_S is the probability that an occurrence of the string begins at a independently selected point in the text, then the occurrences of this string will take up about $-P_S N \log P_S$ amount of storage if each occurrence is encoded as a unit. But if the characters were encoded individually using the underlying Markov based probabilities, the collective occurrence of these characters as contributed by this

string will occupy about $P_S N(-\log P_{s_1} - \cdots - \log P_{s_n s_{n-1}})$ bits, since $-\log P_{s_j s_i}$ would approximate the length of the code for s_i when it follows s_j. But this quantity is identical to the one describing the storage required if we encode the string as a unit. Since the two quantities are equal, no savings result.

Since our objective is to select strings that, when replaced by one byte codewords, will minimize storage requirements, the above analysis suggests that the following criterion should be appropriate: treat a string S as a unit if the savings gained by replacing it by a single byte are large. The criterion for replacing S by a single byte codeword thus becomes: $w_2 = -f_S \log P_S + f_S \log(P_{s_1} P_{s_1 s_2} \cdots) \gg 0$, i.e., $f_S \log \frac{P_{s_1} P_{s_1 s_2} \cdots}{P_S} \gg 0$, where $f_S = P_S N$ is the frequency with which the string occurs. Hence w_2 explicitely incorporates the correlation between the characters forming the string as well as their overall frequency (see also [13]).

5. Clustering

At the end of the first step of the algorithm described in Section 4, we have a sequence of m elements, each a member of an alphabet A, such that the occurrences of these elements are reasonably well described by a first-order Markov process. If we were to continue to the second step directly, we would create tables indicating the probability of an element occuring given the occurrence of the one just scanned. For a higher order Markov process, especially with an extended alphabet, this would create a very large table. We reduce the size of this table by breaking the set of elements we are encoding into clusters. Then, when creating Huffman trees, we use the same value, $P_{s_j s_i}$, for the probability that s_i occurs for all preceding characters, s_j, in the same cluster $\mathcal{C}$ [18]. Thus, if $s_j \in \mathcal{C}$, we could denote this shared cluster based probability by $P_{\mathcal{C} s_i}$. If we have m items and t clusters, we need a table of only tm elements to represent this distribution, instead of the m^2 needed for a first-order Markov process. Our main interest in this paper is in studying the properties of these clusters and to garner some insights about information theory. However, we also believe this to be a practical approach to compression; this belief is encouraged by the results of the example presented in Section 6. The cluster model is represented in Figure 1.

Insert Figure 1 here

Figure 1: Cluster model. Characters are partitioned into clusters $\mathcal{C}_1, \ldots, \mathcal{C}_t$. The probability that c_i occurs depends only on c_i and the cluster with which the preceding character is associated.

Our task then is **1)** to decide how to cluster elements, and **2)** to decide, when creating the Huffman trees, what probability $P_{\mathcal{C}s_i}$ to use for an element s_i contingent on that element following a member of the cluster $\mathcal{C}$. We first deal with the second problem, assuming that the partition of the elements into t non-overlapping clusters $\{\mathcal{C}_1, \ldots, \mathcal{C}_t\}$ is given.

5.1 Cluster probabilities

Suppose that we assign to an arbitrary element, $c_k \in A$, the probability $\hat{P}_{\mathcal{C}k}$ when it appears after a member of the cluster $\mathcal{C}$; we want the optimal value of $\hat{P}_{\mathcal{C}k}$. Thus $\{\hat{P}_{\mathcal{C}k}\}$ is a single probability distribution, approximating the set of distributions $\{P_{ik}\}$, for P_{ik} the true probability of c_k when it follows c_i, for all $c_i \in \mathcal{C}$. Given $\{\hat{P}_{\mathcal{C}k}\}$, we can construct a Huffman tree (or define an interval of appropriate length for arithmetic coding). If c_k follows cluster $\mathcal{C}$, the length of its codeword will be approximately $-\log \hat{P}_{\mathcal{C}k}$.

Thus, if we have just scanned $c_i \in \mathcal{C}$, the average length of the codeword for following element is $-\sum_{k=1}^{m} P_{ik} \log \hat{P}_{\mathcal{C}k}$, and the overall average length of a codeword, averaged over all preceding elements, is

$$H = \sum_{\mathcal{C}} \left[\sum_{c_i \in \mathcal{C}} P_i \left(- \sum_{k=1}^{m} P_{ik} \log \hat{P}_{\mathcal{C}k} \right) \right],$$

where here and below, $\sum_{\mathcal{C}}$ denotes the sum over the clusters $\mathcal{C}$ in $\{\mathcal{C}_1, \ldots, \mathcal{C}_t\}$, and P_i is the unconditional probability of c_i occurring; in a Markov process, this unconditional long-term probability can be computed from the transition matrix [5]. Each term in brackets is associated with a single cluster and depends on a single distribution, $\hat{P}_{\mathcal{C}k}$, which can be changed independently of the others; thus H is minimized if each expression in brackets is minimized. To find the optimal $\{\hat{P}_{\mathcal{C}k}\}$ for a given cluster, we form the Lagrangian

$$\mathcal{L} = - \sum_{c_i \in \mathcal{C}} \sum_{k=1}^{m} P_i P_{ik} \log \hat{P}_{\mathcal{C}k} - \lambda \left(\sum_{k=1}^{m} \hat{P}_{\mathcal{C}k} - 1 \right),$$

and minimize it for values $\hat{P}_{\mathcal{C}k}$ subject to $\sum_k \hat{P}_{\mathcal{C}k} = 1$. We find, for cluster $\mathcal{C}$,

$$\hat{P}_{\mathcal{C}k} = \sum_{c_i \in \mathcal{C}} \frac{P_i P_{ik}}{P_{\mathcal{C}}}, \tag{2a}$$

where

$$P_{\mathcal{C}} = \sum_{c_i \in \mathcal{C}} P_i. \tag{2b}$$

Denote this optimal value for $\hat{P}_{\mathcal{C}k}$ by $P_{\mathcal{C}k}$, which is clearly a probability; it is a weighted average of the probability distributions constituting $\mathcal{C}$. For any cluster $\mathcal{C}$, we shall refer to $\{P_{\mathcal{C}k}\}$ as the probability distribution associated with the cluster.

Note that $P_i/P_{\mathcal{C}}$ is the probability of c_i, given that an element in $\mathcal{C}$ occurred. Since P_{ik} is the probability that c_k will occur, *given* that c_i ($c_i \in \mathcal{C}$) was just scanned, $P_iP_{ik}/P_{\mathcal{C}}$ is the probability that c_i, a element in $\mathcal{C}$, was just scanned *and* c_k follows; summing over i for $c_i \in \mathcal{C}$ gives the average of P_{ik} over i for $c_i \in \mathcal{C}$. Thus $P_{\mathcal{C}k}$ is interpretable as the probability of c_k, given that some element in $\mathcal{C}$ was just scanned.

We can now write H as

$$H = \sum_{\mathcal{C}} P_{\mathcal{C}}(-\sum_{k} P_{\mathcal{C}k} \log P_{\mathcal{C}k}) \equiv \sum_{\mathcal{C}} P_{\mathcal{C}} H_{\mathcal{C}}, \tag{2c}$$

with

$$H_{\mathcal{C}} = -\sum_{k} P_{\mathcal{C}k} \log P_{\mathcal{C}k} \tag{2d}$$

the entropy defined by the cluster based probability. $P_{\mathcal{C}k}$ is the "average" probability within the cluster $\mathcal{C}$; below, when we want to emphasize this fact we will adopt the notation $H_{\bar{\mathcal{C}}}$ for $H_{\mathcal{C}}$. $H_{\mathcal{C}}$ is the "ideal length" of the encoding of an element following an element in $\mathcal{C}$. Our task then is, given a value t, to find a partition of A into t clusters, such that H is minimized.

Two special cases are particularly interesting:

(a) If $t = 1$, then we are treating the entire alphabet as a single element. Then $P_{\mathcal{C}}$ is 1, and $P_{\mathcal{C}k}$ is simply the *a priori* probability of c_k — the value we would use if we ignored the Markov property.

(b) If $\mathcal{C}$ is a single element, $\{c_i\}$, then $P_{\mathcal{C}} = P_i$ and $P_{\mathcal{C}k} = P_{ik}$. Now $H_{\mathcal{C}}$ is in effect H_i, the single character entropy. If each element is in its own cluster ($t = m$), then we have a full Markov model.

5.2 Clustering loss function

The task of finding the optimal partition is likely to be very difficult. Indeed, very similar problems have been found to be NP-complete. We instead search for heuristics that are reasonable. A straightforward and often effective approach is to adopt a greedy algorithm, beginning with the individual elements as elementary clusters and at each stage merging several clusters. We define the loss, $\mathcal{L}$, in average storage required per element due to merging clusters into *superclusters* as $\mathcal{L} = H_2 - H_1$, where the indices

on H distinguish the entropy of the original partition (H_1) from that of the new one (H_2). We will usually omit from $\mathcal{L}$ the subscripts that define the partitions merged to create $\mathcal{L}$ since it is generally clear which clusters are involved.

$\mathcal{L}$ can be reexpressed in several useful ways:

(a) Since the original partition is a refinement of the new, merged partition, $\mathcal{L}$ can be naturally decomposed into components, each associated with one new cluster, and the analysis carried out separately on each; this expresses $\mathcal{L}$ in terms of losses associated with each cluster. Consider, then, the contribution to $\mathcal{L}$ of disjoint clusters $\{\mathcal{C}_r\}$ combining to form $\mathcal{C}$. To make this decomposition explicit, $\mathcal{L}$ can be written as

$$\mathcal{L} = \sum_{\mathcal{C}} P_{\mathcal{C}} (H_{\mathcal{C}} - \sum_{\mathcal{C}_r \subseteq \mathcal{C}} \frac{P_{\mathcal{C}_r}}{P_{\mathcal{C}}} H_{\mathcal{C}_r}) \equiv \sum_{\mathcal{C}} P_{\mathcal{C}} L_{\mathcal{C}}, \tag{3a}$$

where

$$L_{\mathcal{C}} = H_{\mathcal{C}} - \sum_{\mathcal{C}_r \subseteq \mathcal{C}} \frac{P_{\mathcal{C}_r}}{P_{\mathcal{C}}} H_{\mathcal{C}_r} \tag{3b}$$

is the loss associated with $\mathcal{C}$. The overall, weighted loss, $\mathcal{L}$, is the average of the unweighted losses incurred when forming the individual superclusters. If we are simply combining several clusters into a single cluster, $\mathcal{C}$, then (3a) becomes

$$\mathcal{L} = P_{\mathcal{C}} L_{\mathcal{C}}, \tag{3c}$$

with the terms associated with the unmodified clusters cancelling.

(b) We can rewrite $L_{\mathcal{C}}$ more evocatively as

$$L_{\mathcal{C}} = H_{\bar{\mathcal{C}}} - \bar{H}_{\mathcal{C}}, \tag{4}$$

where $H_{\bar{\mathcal{C}}} = H_{\mathcal{C}}$ and $\bar{H}_{\mathcal{C}} = \sum_{\mathcal{C}_r \subseteq \mathcal{C}} \frac{P_{\mathcal{C}_r}}{P_{\mathcal{C}}} H_{\mathcal{C}_r} = \sum_{\mathcal{C}_r \subseteq \mathcal{C}} \frac{P_{\mathcal{C}_r}}{P_{\mathcal{C}}} \left(- \sum_k P_{\mathcal{C}_r k} \log P_{\mathcal{C}_r k} \right)$: that is, $L_{\mathcal{C}}$ is the difference between the entropy of the average probability distribution in the cluster and the average of the individual entropies of the clusters comprising it. Ultimately, a cluster is made up of individual probability distributions. If $P_{\mathcal{C}_r} = P_i$ and $P_{\mathcal{C}_r k} = P_{ik}$, equation (4) describes the loss of merging a number of elementary distributions into a cluster.

(c) We would also like a representation of $\mathcal{L}$ directly in terms of the basic cluster probabilities. Expanding $H_{\mathcal{C}}$ and $H_{\mathcal{C}_r}$, $P_{\mathcal{C}} L_{\mathcal{C}}$ can be rewritten as

$$P_{\mathcal{C}} H_{\mathcal{C}} - \left(\sum_{\mathcal{C}_r \subseteq \mathcal{C}} P_{\mathcal{C}_r} H_{\mathcal{C}_r} \right) = \left(-P_{\mathcal{C}} \sum_k P_{\mathcal{C}k} \log P_{\mathcal{C}k} \right) - \sum_{\mathcal{C}_r \subseteq \mathcal{C}} P_{\mathcal{C}_r} \left(- \sum_k P_{\mathcal{C}_r k} \log P_{\mathcal{C}_r k} \right).$$

But $P_{\mathcal{C}} P_{\mathcal{C}k} = \sum_{c_i \in \mathcal{C}} P_i P_{ik} = \sum_{\mathcal{C}_r \subseteq \mathcal{C}} \left(\sum_{c_i \in \mathcal{C}_r} P_i P_{ik} \right) = \sum_{\mathcal{C}_r \subseteq \mathcal{C}} P_{\mathcal{C}_r} P_{\mathcal{C}_r k}$, so

$$\begin{aligned} P_{\mathcal{C}} L_{\mathcal{C}} &= \sum_{\mathcal{C}_r \subseteq \mathcal{C}} P_{\mathcal{C}_r} \left(-\sum_k P_{\mathcal{C}_r k} \log P_{\mathcal{C}k} \right) - \sum_{\mathcal{C}_r \subseteq \mathcal{C}} \left(-P_{\mathcal{C}_r} \sum_k P_{\mathcal{C}_r k} \log P_{\mathcal{C}_r k} \right) \\ &= \sum_{\mathcal{C}_r \subseteq \mathcal{C}} P_{\mathcal{C}_r} \left(-\sum_k P_{\mathcal{C}_r k} \log \frac{P_{\mathcal{C}k}}{P_{\mathcal{C}_r k}} \right) \end{aligned}$$

and $\mathcal{L}$ is the sum of these values:

$$\mathcal{L} = \sum_{\mathcal{C}} \sum_{\mathcal{C}_r \subseteq \mathcal{C}} P_{\mathcal{C}_r} \left(-\sum_k P_{\mathcal{C}_r k} \log \frac{P_{\mathcal{C}k}}{P_{\mathcal{C}_r k}} \right). \tag{5a}$$

Thus we get from the definition of $L_{\mathcal{C}}$ in terms of $\mathcal{L}$ (eq. (3a))

$$L_{\mathcal{C}} = \sum_{\mathcal{C}_r \subseteq \mathcal{C}} \frac{P_{\mathcal{C}_r}}{P_{\mathcal{C}}} \left(-\sum_k P_{\mathcal{C}_r k} \log \frac{P_{\mathcal{C}k}}{P_{\mathcal{C}_r k}} \right). \tag{5b}$$

5.2.1 Interpretation of loss function

We can now make the following observations:

First note that $L_{\mathcal{C}}$ is a weighted average of terms of the form $-\sum P_k \log \frac{Q_k}{P_k}$, with $\{P_k\}$ and $\{Q_k\}$ probabilities. As shown above (eq. (1)), this sum is greater than zero unless $P_k \equiv Q_k$, in which case the sum is zero. Thus $L_{\mathcal{C}}$ will be zero if and only if $P_{\mathcal{C}k} = P_{\mathcal{C}_r k}$ for all k, and each $\mathcal{C}_r$ that is being merged into $\mathcal{C}$. Otherwise $L_{\mathcal{C}} > 0$, i.e., the "loss" is a genuine loss — the average length of the code resulting from a merging of clusters does increase, unless all the probabilities being merged into one cluster are identical. The closer $L_{\mathcal{C}}$ is to zero, the better the clustering.

Second, note that the term in parenthesis in equation (5b) has the form $-\sum P_{true} \log \frac{P_{approx}}{P_{true}}$, where a cluster based distribution is approximating the component distributions comprising it. This formula has previously been used as a measure of how well an approximate probability distribution agrees with the true distribution it is estimating (see, e.g., [16]). The appearance of this measure here, motivated by compression considerations, suggests an easily understandable, intuitively satisfying, interpretation: given a probability distribution $\{P_i\}$ and an approximation to that distribution $\{Q_i\}$, we can use as a measure of the goodness of the approximation the expected deterioration in code length of using the approximate distribution as a substitute for the true distribution when compressing the data.

The interpretation of the above approximation formula is important enough to merit a more detailed argument. As we saw above, if we approximate P_k, the probability associated with the character c_k, by Q_k, the length of the encoding of c_k will be about $-\log Q_k$ and the expected length of the code based on Q_k will be about $-\sum P_k \log Q_k$. The optimal length is $-\sum P_k \log P_k$. Thus the expected deterioration is $-\sum P_k \log \frac{Q_k}{P_k}$. In our problem, P_k is the true probability of an element c_k occurring, conditional on having scanned a specific element, say c_i. Q_k is our cluster based approximation of P_k. In general, any set of probabilities and approximations to them can be interpreted in this manner. The interpretation of the measure as an increase in expected coding size is concrete and easily understandable, and provides an alternative to the more abstract idea of "information loss".

In constructing $L_{\mathcal{C}}$, a number of distributions, $\{P_{\mathcal{C}_r k}\}$, are approximated by $P_{\mathcal{C}k}$. The weighted average as given above generalizes the formula measuring how well one distribution estimates another: $L_{\mathcal{C}}$ estimates how well one distribution estimates a set of distributions.

5.2.2 Special cases

It is instructive to note explicitely the form taken by the loss function for a few special cases.

(a) We first note the special case in which each $\mathcal{C}_r$ is an isolated element: $\mathcal{C}_r = \{c_r\}$. Then $P_{\mathcal{C}_r k} = P_{rk}$, $L_{\mathcal{C}} = H_{\mathcal{C}} - \sum_r \frac{P_r}{P_{\mathcal{C}}} H_r$, and H_r is $-\sum_k P_{rk} \log P_{rk}$, the entropy associated with first having scanned c_r. Expanding, we find

$$L_{\mathcal{C}} = \left(-\sum_k P_{\mathcal{C}k} \log P_{\mathcal{C}k} \right) - \sum_{c_r \in \mathcal{C}} \frac{P_r}{P_{\mathcal{C}}} \left(-\sum_k P_{rk} \log P_{rk} \right).$$

Thus the cummulative loss contributed by a cluster is the length associated with the cluster minus the average of the lengths associated with the elements making up the cluster. This quantity is interesting because it gives us the overall deterioration due to the clustering of elements at any given stage. As such, $L_{\mathcal{C}}$ can be interpreted as a general measure of the lack of cohesiveness of a set of probability distributions. Note that if for all k, $P_{ik} = P_{jk}$ for $c_i \neq c_j$ in $\mathcal{C}$, then $L_{\mathcal{C}} = 0$, and $L_{\mathcal{C}}$ always is greater than or equal to zero. Also, $L_{\mathcal{C}}$ is symmetric over $c_i \in \mathcal{C}$.

(b) A second interesting case is when all of the clusters merge into a single cluster, the entire alphabet A; this quantity is a measure of "headroom": how much capacity

we still have for loss as we continue clustering. This is given by $L = H_A - \sum P_{\mathcal{C}} H_{\mathcal{C}}$, with $H_A = -\sum_{k=1}^{m} P_k \log P_k$.

(c) Another interesting measure is $L = H_A - \sum_{i=1}^{m} P_i H_i$, with H_A as before and P_k the unconditional probability of c_k occuring: L is then the loss ignoring the Markov structure of the generator. This measure indicates whether we should consider the Markov model at all, since H_A is the average length if we created our code under the assumption of independent character generation. This argument can be pursued further, by asking what is the information content of a Markov process [12], [8]. Consider a string of n characters generated by the Markov process. The probability of the string $S = s_1 s_2 \cdots s_n$ is $P_S = P_{s_1} P_{s_1 s_2} \cdots P_{s_{n-1} s_n}$, and the information content of the generator is $H = -\sum_S P_S \log P_S$. Expanding the logarithm and collecting terms, this becomes $H = H_0 + (n-1)\bar{H}_A$, where $H_0 = -\sum P_i \log P_i$ and $\bar{H}_A = \sum_i P_i(-\sum_k P_{ik} \log P_{ik}) = \sum_i P_i H_i$, i.e., $\bar{H}_A$ denotes $\bar{H}_{\mathcal{C}}$, where the cluster $\mathcal{C}$ is the entire alphabet A.

Thus the information content per character, H/n, for large n, approaches $\bar{H}_A$. The inequality $L_A \geq 0$ or (from (4)) $H_{\bar{A}} \geq \bar{H}_A$, tells us that if the generator is Markov, we can only improve compression by recognizing this. More generally, if we have a Markov process, recognizing this gives us information about the strings that are being generated. $\bar{H}_A$ is the information content in the string once we recognize the string as being generated by a Markov process. $H_{\bar{A}} - \bar{H}_A$ is, in effect, the information conveyed to us, when being told that the string was created by a Markov generator. Alternatively, when a message is generated by a stochastic device with structure, recognizing that structure conveys information about the strings that result, and this information can be used to reduce the information content of the string, quantified as the number of bits needed to store it.

(d) Finally, we consider the effect of merging two clusters into one; this will form the basis of the algorithm proposed in Section 5.3. Suppose then that the disjoint clusters $\mathcal{C}_1$ and $\mathcal{C}_2$ are merged to form $\mathcal{C}$. Equation (3a) becomes $\mathcal{L} = P_{\mathcal{C}} L_{\mathcal{C}}$, with $L_{\mathcal{C}} = H_{\mathcal{C}} - \left(\frac{P_{\mathcal{C}_1}}{P_{\mathcal{C}}} H_{\mathcal{C}_1} + \frac{P_{\mathcal{C}_2}}{P_{\mathcal{C}}} H_{\mathcal{C}_2} \right)$, and since it is easy to see that $P_{\mathcal{C}k} = \frac{P_{\mathcal{C}_1} P_{\mathcal{C}_1 k} + P_{\mathcal{C}_2} P_{\mathcal{C}_2 k}}{P_{\mathcal{C}_1} + P_{\mathcal{C}_2}}$, we get the more explicit form

$$\begin{aligned} L_{\mathcal{C}} = &- \sum_{k=1}^{m} \frac{P_{\mathcal{C}_1} P_{\mathcal{C}_1 k} + P_{\mathcal{C}_2} P_{\mathcal{C}_2 k}}{P_{\mathcal{C}_1} + P_{\mathcal{C}_2}} \log \frac{P_{\mathcal{C}_1} P_{\mathcal{C}_1 k} + P_{\mathcal{C}_2} P_{\mathcal{C}_2 k}}{P_{\mathcal{C}_1} + P_{\mathcal{C}_2}} \\ &+ \frac{P_{\mathcal{C}_1}}{P_{\mathcal{C}_1} + P_{\mathcal{C}_2}} \sum_{k=1}^{m} P_{\mathcal{C}_1 k} \log P_{\mathcal{C}_1 k} + \frac{P_{\mathcal{C}_2}}{P_{\mathcal{C}_1} + P_{\mathcal{C}_2}} \sum_{k=1}^{m} P_{\mathcal{C}_2 k} \log P_{\mathcal{C}_2 k}. \end{aligned} \tag{6}$$

$L_{\mathcal{C}}$ is a measure of dissimilarity of two clusters, and, more generally, a weighted measure of the dissimilarity of two probability distributions: in common with other measures of dissimilarity, it takes positive values, is symmetric in the clusters, and is equal to zero if and only if $\{P_{\mathcal{C}_1 k}\} = \{P_{\mathcal{C}_2 k}\}$ and thus equal to $\{P_{\mathcal{C}k}\}$. In general, if we wish to compute a loss for combining two probability distributions, but do not have explicit values for $P_{\mathcal{C}_1}$ and $P_{\mathcal{C}_2}$, we can set both equal to 1/2.

5.3 Clustering heuristic

This suggests a heuristic for creating the clusters: beginning with the individual elements as primary clusters, we iteratively combine pairs of clusters. At each stage, we combine $\mathcal{C}_r$ and $\mathcal{C}_s$ to form cluster $\mathcal{C}_{rs}$ provided that $\mathcal{L}_{rs} \equiv P_{\mathcal{C}_{rs}} L_{\mathcal{C}_{rs}}$, the loss after combination, is less than that for any other pair of clusters. Thus $\mathcal{L}_{rs} \leq \mathcal{L}_{uv}$, for u, v denoting any other pair of clusters that are candidates for combination. Note that the critical value determining whether to combine two clusters is the product of the closeness of the two clusters and the likelihood of an element of these clusters occurring. Thus we may well find ourselves combining quite different clusters if their elements occur rarely.

Clustering procedures [25] often begin by creating a measure of similarity, and then continue by somehow combining items using this measure. Although the measure of similarity is a critical component of this process, it tends to be chosen on an *ad hoc* basis. Our clustering procedure is unusual in being based on a measure of association that itself was directly developed out of our objectives for creating clusters. The following procedure is therefore used (repeating the required equations, for the convenience of the reader):

1. *Initialization:* For each element c_i (treated as a primary cluster), store $P_{\mathcal{C}_i} = P_i$, $P_{\mathcal{C}_i k} = P_{ik}$ and $H_i = -\sum_k P_{ik} \log P_{ik}$.
2. *Iteration:* At each stage, compute $\mathcal{L}_{rs}$ $(r < s)$ for each pair of clusters $(\mathcal{C}_r, \mathcal{C}_s)$:
   (a) $P_{\mathcal{C}_{rs}} = P_{\mathcal{C}_r} + P_{\mathcal{C}_s}$ (eq. (2b))
   (b) $P_{\mathcal{C}_{rs}k} = (P_{\mathcal{C}_r} P_{\mathcal{C}_r k} + P_{\mathcal{C}_s} P_{\mathcal{C}_s k})/P_{\mathcal{C}_{rs}}$ (eq. (2a))
   (c) $H_{rs} = -\sum_{k=1}^{m} P_{\mathcal{C}_{rs}k} \log P_{\mathcal{C}_{rs}k}$ (eq. (2d))
   (d) $L_{\mathcal{C}_{rs}} = H_{rs} - \left(\frac{P_{\mathcal{C}_r}}{P_{\mathcal{C}_{rs}}} H_{\mathcal{C}_r} + \frac{P_{\mathcal{C}_s}}{P_{\mathcal{C}_{rs}}} H_{\mathcal{C}_s} \right)$ (eq. (3b))
   (e) $\mathcal{L}_{rs} = P_{\mathcal{C}_{rs}} L_{\mathcal{C}_{rs}}$ (eq. (3c))

   After the initial stage, $\mathcal{L}_{rs}$ need be computed only between the new cluster and

those older clusters remaining after the merged clusters are removed.

3. Either combine the two clusters that yield the smallest value for $\mathcal{L}_{rs}$ or stop if adequate clustering has taken place. As our stopping criterion, we could use a threshold on the loss function, stopping when the cost exceeds this threshold; another possibility is to continue until the set of clusters has been reduced to a predetermined number.

4. If clusters $\mathcal{C}_r$ and $\mathcal{C}_s$ are combined to form cluster $\mathcal{C}_{rs} \equiv \mathcal{C}$:

   (a) remove clusters $\mathcal{C}_r$ and $\mathcal{C}_s$ from consideration;

   (b) enter cluster $\mathcal{C}$;

   (c) associate with $\mathcal{C}$ the values $P_{\mathcal{C}}$, $P_{\mathcal{C}k}$ and $H_{\mathcal{C}}$ as computed in 2.(a) 2.(b) and 2.(c);

5. Return to step 2.

Each iteration reduces the number of clusters by one. The matrix of $\mathcal{L}$'s for cluster pairs must be updated only for pairs involving the new cluster. We keep track of the partition of the primary elements into clusters using well-known Union-Find algorithms (see [1]). Our greedy algorithm is not necessarily optimal, but should produce reasonable results. The end structure permits us to calculate $L_{\mathcal{C}}$ and $\mathcal{L}$.

6. Example

In this section, we work through a detailed example. We wanted our example to be manageably small in the size of both the text and the alphabet, yet not to be completely artificial. Both goals are met by using music as our text source. We chose the *Sonata in C major for Flute and Basso continuo, BWV 1033*, by Johann Sebastian Bach, consisting of five movements with a total of 1180 notes. For simplicity, the notes were considered modulo an octave, and sharps and flats were ignored—that is, the "alphabet" consists of the seven notes in the scale of C: $\{C, D, E, F, G, A, B\}$. Also, the first note of each movement was used only for computing the first-order Markov transition probabilities of the following notes, but were not counted themselves as belonging to the "text", leaving a text of 1175 "characters" to be compressed. As a baseline, we use the space required by fixed length encoding. Since we have seven characters, we need $\lceil \log_2 7 \rceil = 3$ bits per character, or 3525 bits all told.

Table 1: *First-order Markov probabilities*

	A	B	C	D	E	F	G	P^0	N
A	0.0124	0.2795	0.1553	0.0683	0.0311	0.1056	0.3478	0.1370	161
B	0.3421	0.0132	0.3750	0.0855	0.0395	0.0461	0.0987	0.1294	152
C	0.1139	0.3218	0.0495	0.2376	0.1337	0.0545	0.0891	0.1719	202
D	0.0180	0.0958	0.3533	0.0659	0.2934	0.0838	0.0898	0.1421	167
E	0.0983	0.0173	0.1618	0.3006	0.0173	0.3121	0.0925	0.1472	173
F	0.1032	0.0129	0.0516	0.1484	0.3677	0.0387	0.2774	0.1319	155
G	0.3030	0.1152	0.0970	0.0545	0.1333	0.2788	0.0182	0.1404	165

Table 1 summarizes the statistical characteristics of our text. The first seven columns of each row of Table 1 contain the conditional probabilities of a character occurring, given that the character defining the row has just been observed. Row i of the column labelled P^0 gives the unconditional probability of character i occurring. The column labelled N gives the total number of occurrences of the row character.

(a) *Simple Huffman code:* If we don't recognize the Markov property, we could construct a straightforward Huffman code based on the unconditional probabilities P_i^0. The unconditional distribution of the seven characters is suprisingly uniform. The corresponding Huffman code is almost a fixed length code.

Any Huffman code can be described by a string of integers, $\langle n_1, \ldots, n_\ell \rangle$, where n_i, for $1 \le i \le \ell$, is the number of codewords of length i bits, and ℓ is the length of the longest codeword (the depth of the tree). Note that $\sum_{i=1}^{\ell} n_i$ is the size of the alphabet, in our case 7, and that $\sum_{i=1}^{\ell} n_i 2^{-i} = 1$, a property of all Huffman codes. For the unconditional distribution, the Huffman code can be described in this manner by the string $\langle 0, 1, 6 \rangle$: there are no codewords of length 1, a single codeword of length 2 and six codewords of length 3. This encoding uses 3323 bits to encode the entire text, or 2.828 bits per character on the average. Comparing this to a fixed length code, we find that simple Huffman coding yields 5.7% compression. This modest result is due to the uniformity of the distribution.

(b) *Markov model code:* At the opposite extreme, we can treat the text as having been generated by a Markov process and encode a character using the Huffman tree derived from the probability distribution associated with the preceding character. The results of such an encoding are presented in Table 2.

The first column of Table 2 gives a description of the Huffman tree for each of the conditional probability distributions. The second column gives the average codeword

length (ACL) of the characters defining the rows: that is, the average codeword length as given in row i is the average number of bits needed to encode a character following an instance of character i. Finally, the last column of Table 2 gives the entropies of the conditional distributions of the corresponding rows in Table 1.

Table 2: *Statistical information on conditional probability distributions*

	Huffman code	ACL	Entropy
A	$\langle 0,3,1,1,2\rangle$	2.373	2.302
B	$\langle 1,1,0,3,2\rangle$	2.243	2.164
C	$\langle 0,2,3,2\rangle$	2.545	2.518
D	$\langle 0,2,3,2\rangle$	2.437	2.348
E	$\langle 0,3,1,1,2\rangle$	2.387	2.320
F	$\langle 1,1,1,1,1,2\rangle$	2.348	2.274
G	$\langle 0,2,3,2\rangle$	2.491	2.443

The conditional probabilities are much more skewed than the unconditional distribution. This is evident from Table 1 and also from the forms of the corresponding Huffman trees (first column of Table 2). We find that one needs 2832 bits to encode the text as a first-order Markov process, or 2.410 bits per character on the average, a 19.7% compression gain over the baseline fixed length code. The more than three-fold improvement in compression over the simple Huffman model is due to the emergence of skewed distributions when conditional probabilities are considered — this skewing disappears when all the distributions are merged. For this example, the recognition of the Markov property yields therefore a significant improvement.

The last two columns of Table 2 show that the ACL's are in fact quite close to the entropy, the theoretical lower bound: on our example, the Huffman codes are between 1% and 4% longer. Our decision to use the entropy, rather than the actual length, in our compression heuristic is experimentally justified by the fact that both entropy and ACL induce the same ordering on the given set of distributions: sorting the rows by decreasing values of either ACL or the entropy yields the permutation $CGDEAFB$.

Our example is too small to allow the replacement of some highly correlated strings by a new symbol, as suggested in Section 4. If we had chosen a piece of music in some other key, there would be notes which would almost always be preceded by a sharp or flat sign. Such pairs, (sharp, note) or (flat, note), would probably be good candidates for substitution. Similarly, runs and chords would be interesting possibilities. For our example, we concentrate on the clustering proposed in Section 5.

(c) *Cluster-based code:* Intermediate between procedures **(a)** and **(b)** is basing the compression on the probability distributions associated with clusters of characters. To do this, we first must compute the table of losses, $\mathcal{L}$. Table 3 contains this information as well as the contents of the subsequent loss tables. The order of the characters in the rows and columns of Table 3 has been chosen so as to permit us to encompass all this information in a single table. The loss table corresponding to the initial state (7 clusters, each consisting of a singleton) is the sub-matrix bounded by rows D and G and by columns F and E. The cluster with minimum loss (0.054) is DF. To compute this loss, we first find $P_{DF} = P_D + P_F = 0.1421 + 0.1319 = 0.2740$. We then compute the probability distribution $P_{DF,k} = (0.0590,\ 0.0559,\ 0.2081,\ 0.1056,\ 0.3292,\ 0.0621,\ 0.1801)$. For example, the $\mathcal{C}_{DF}$ cluster probability of A is $(0.1421 \times 0.0180 + 0.1319 \times 0.1032)\ /\ 0.2740 = 0.0590$. The entropy for the $\mathcal{C}_{DF}$ cluster, based on the probability distribution $P_{DF,k}$, is $-(0.0590 \log_2 0.0590 + \cdots) = 2.509$; the average entropy of the components is $\frac{0.1421}{0.2740} \times 2.348 + \frac{0.1319}{0.2740} \times 2.274 = 2.312$. The weighted loss, $\mathcal{L}_{DF}$, is thus $0.2740 \times (2.509 - 2.312)$, or 0.054, as appears in the table.

Table 3: *Clustering loss matrix*

	F	A	C	B	G	E	DF	AC	BG	EBG
D	0.054	0.059	0.072	0.065	0.070	0.083				
F		0.082	0.063	0.078	0.083	0.085				
A			0.055	0.091	0.093	0.080	0.073			
C				0.094	0.061	0.080	0.077			
B					0.056	0.058	0.074	0.104		
G						0.057	0.088	0.086		
E							0.093	0.086	0.059	
DF								0.082	0.093	0.113
AC									0.116	0.127

Note that requiring $L_{\mathcal{C}}$, the unweighted loss associated with a cluster, to be multiplied by $P_{\mathcal{C}}$, the probability of the cluster, was critical in the choice of DF: the loss of DF before multiplication by $P_{\mathcal{C}}$ was $2.509 - 2.312 = 0.197$, which was only second smallest, following the corresponding quantity for AC, which was 0.178.

The next step in the clustering algorithm is to eliminate the rows and columns corresponding to D and F and to create a new column for the cluster DF. The resulting loss table is the sub-matrix of Table 3 bounded by rows A and E and by columns C and DF. The minimum in this sub-matrix is $\mathcal{L}_{AC} = 0.055$, so the next cluster formed is AC. The resulting loss matrix is the sub-matrix bounded by rows

B and DF and by columns G and AC. The minimum is $\mathcal{L}_{BG} = 0.056$, therefore the next cluster to form is BG, with loss matrix given by the sub-matrix bounded by rows E and AC and by columns DF and BG. The minimum is now $\mathcal{L}_{EBG} = 0.059$, so EBG is created. At this stage, the loss matrix contains only 3 elements: column EBG and the element at the intersection of row DF and column AC, 0.082; the latter is the smallest of the three. Thus the final cluster to be formed is $ACDF$.

Table 4 and Figure 2 summarize the results. Each line of Table 4 corresponds to one iteration, i.e., the creation of a new cluster; the first line corresponds to the initial, full first-order Markov model, and the last line to the model of independent character generation. The values in the column labelled compression are the percent reductions in storage compared to the fixed length baseline model. For each newly formed cluster, the following information is given: a characterization of the corresponding Huffman code; the partition of the alphabet into clusters at this stage of the heuristic; the new total length of the encoding; and, in the last column, the overall compression obtained using the given partition. Figure 2 represents this information graphically.

Table 4: *Summary of clustering heuristic*

new cluster	Huffman code	partition	total length	compression
		$A\ B\ C\ D\ E\ F\ G$	2832	19.7%
DF	$\langle 0,3,0,4\rangle$	$A\ B\ C\ E\ G\ \ DF$	2887	18.1%
AC	$\langle 0,2,3,2\rangle$	$B\ E\ G\ \ DF\ \ AC$	2949	16.3%
BG	$\langle 0,3,0,4\rangle$	$E\ \ DF\ \ AC\ \ BG$	3009	14.6%
BGE	$\langle 0,3,1,1,2\rangle$	$DF\ \ AC\ \ BGE$	3071	12.9%
$ACDF$	$\langle 0,2,3,2\rangle$	$BGE\ \ ACDF$	3164	10.2%
$ACDFBGE$	$\langle 0,1,6\rangle$	$ACDFBGE$	3323	5.7%

Insert Figure 2 here

Figure 2: Clustering dendogram. This diagram indicates, as we go from the bottom up, the formation of clusters. The vertical level at which two clusters merge represents the number of bits needed to store the text using that cluster representation. The implied savings in space over the baseline case is given in parenthesis.

The first three clusters formed are of pairs of notes a third (two notes) apart. The statistical reason for this is that both notes are strongly associated with the intermediate note — for example, the probability of E occurring is high if either a D or an F has just occurred.

The results exhibit the internal-space/compression tradeoff: the more clusters we use, the more internal space we need for storing the different Huffman trees, but the less we need for encoding the text and, therefore, the better the compression. In this example, recognizing just a few clusters enables us to realize much of the advantages of using a Markov model; this result should be tested on larger samples of text.

7. Conclusion

It has often been observed that Shannon's theory of information offers important insights for data compression. In this paper, we showed that this favor can in part be returned, and that the possibility of compression, coupled with a model of text generation, permits an independent derivation of the Shannon entropy measure. Possibly, other aspects of a complete information theory can also be developed from this perspective. That information theory is a fertile generator of ideas valuable for compression, however, is reaffirmed in this paper. It provides a natural measure of association of two probability distributions, and makes possible the clustering algorithm that forms the basis of the compression approach suggested above.

References

[1] **Aho A.V., Hopcroft J.E., Ullman J.D.,** *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, MA (1974).

[2] **Ash R.**, *Information Theory*, John Wiley & Sons, New York (1965).

[3] **Cleary J.G., Witten I.H.,** Data compression using adaptive coding and partial string matching, *IEEE Trans. on Communications,* **COM–32** (1984) 396–402.

[4] **Even S.,** *Graph Algorithms*, Computer Science Press (1979).

[5] **Feller W.,** *An Introduction to Probability Theory and Its Applications*, Vol **I**, John Wiley & Sons, Inc., New York (1950).

[6] **Fraenkel A.S., Mor M., Perl Y.,** Is text compression by prefixes and suffixes practical? *Acta Informatica* **20** (1983) 371–389.

[7] **Hadley, G.,** *Nonlinear and Dynamic Programming*, Addison-Wesley, Reading, Mass. (1964).

[8] **Hamming R.W.,** *Coding and Information Theory*, second edition, Prentice-Hall, Englewood Cliffs, NJ (1986).

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

[10] **Hopcroft J.E., Ullman J.D.,** *Introduction to Automata Theory, Languages and Computation,* Addison-Wesley, Reading, MA (1979).

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

[12] **Khinchin A.,** *Mathematical Foundations of Information Theory,* Dover, New York (1957).

[13] **Klein S.T., Bookstein A., Deerwester S.,** Storing text retrieval systems on CD-ROM: compression and encryption considerations, *ACM Trans. on Information Systems* **7** (1989) 230–245.

[14] **Knuth D.E.,** *The Art of Computer Programming, Vol I, Fundamental algorithms*, Addison-Wesley, Reading, Mass. (1973).

[15] **Lelewer D.A., Hirschberg D.S.,** Data Compression, *ACM Computing Surveys* **19** (1987) 261–296.

[16] **Lewis II, P.M.,** Approximating probability distributions to reduce storage requirements, *Information and Control* **2** (1959) 214–225.

[17] **Llewellyn J.A.,** Data compression for a source with Markov characteristics, *The Computer Journal,* **30** (1987) 149–156.

[18] **Loh L.S., Mommens J.H., Raviv J.,** Method of achieving data compaction utilizing variable-length dependent coding techniques, US Patent 3694813 (1972).

[19] **Longo G., Galasso G.**, An application of informational divergence to Huffman codes, *IEEE Trans. on Inf. Th.* **IT–28** (1982) 36–43.

[20] **Ramabadran T.V., Cohn D.L.,** An adaptive algorithm for the compression of computer data, *IEEE Trans. on Communications,* **COM–37** (1989) 317–324.

[21] **Rissanen J., Langdon G.G.,** Arithmetic coding, *IBM J. Res. Dev.* **23** (1979) 149–162.

[22] **Rissanen J., Langdon G.G.,** Universal modeling and coding, *IEEE Trans. on Inf. Th.* **IT–27** (1981) 12–23.

[23] **Rubin F.,** Experiments in text file compression, *Comm. ACM* **19** (1976) 617–623.

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

[25] **Sneath P.H.A., Sokal R.R.,** *Numerical Taxonomy, The Principles and Practice of Numerical Classification,* W.H. Freeman and Company, San Francisco (1973).

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

[27] **Walker V.R.,** Compaction of names by x-grams, *Proc. ASIS* Vol **6** (1969) 129–133.

[28] **Witten I.H., Neal R.M., Cleary J.G.,** Arithmetic coding for data compression, *Comm. ACM* **30** (1987) 520–540.

[29] **Ziv J., Lempel A.,** Compression of individual sequences via variable-rate coding, *IEEE Trans. on Inf. Th.* **IT–24** (1978) 530–536.