Modeling Word Occurences for the Compression of Concordances*

A. Bookstein
Center for Information and Language Studies
University of Chicago
Chicago, IL 60637
bkst@caper.uchicago.edu

S. T. Klein
Dept. of Math. & CS
Bar Ilan University
Ramat-Gan 52900
Israel
tomi@bimacs.cs.biu.ac.il

T. Raita
Comp. Sci. Dept.
University of Turku
20520 Turku
Finland
raita@euroni.cs.utu.fi

Abstract

An earlier paper developed a procedure for compressing concordances, assuming that all elements occurred independently. The models introduced in that paper are extended here to take the possibility of clustering into account. The concordance is conceptualized as a set of bitmaps, in which the bit locations represent documents, and the 1-bits represent the occurrence of given terms. Hidden Markov models (HMM) are used to describe the clustering of the 1-bits. However, for computational reasons, the HMM is approximated by traditional Markov models. A set of criteria is developed to constrain the allowable set of n-state models, and a full inventory is given for $n \leq 4$. Graph theoretic *reduction* and *complementation* operations are defined among the various models, and are used to provide a structure relating the models studied. Finally, the new methods were tested on the concordances of the English Bible and of two of the world's largest full-text retrieval system: the Trésor de la Langue Française and the Responsa Project.

1. Introduction

With increasing attention being given to the *digital library*—massive amounts of information, much of it in textual format, stored and widely accessible by computer— data compression is assuming a new degree of importance. As noted before [16], effective compression of a text-based information retrieval system (IRS) involves far more than compressing the text itself: it is often overlooked that to be able to access and manipulate text, auxiliary data-structures must also be created and stored.

Most large information retrieval systems depend on inverted files for access to their information. In this approach, query processing does not directly involve the original text files (in which key words might be located using some pattern matching technique), but rather auxiliary *dictionary* and *concordance* files. The dictionary is a list of all the different words appearing in the text and is usually ordered alphabetically. For each entry in the dictionary, there is a pointer into the concordance, which lists each occurrence of the word. While the dictionary is only moderately large, the exact size of a concordance depends on a number of parameters, such as the omission or inclusion of the most frequent words (the so-called *stop-words*) and whether stemming is first done — in our experiments, all words, as they appear in the text, are used. However, the size of an uncompressed concordance is generally of the same order of magnitude of that of the text itself.

*Two of the authors (A.B. and S.T.K.) wish to acknowledge that the material in this paper is based upon research supported by the U.S. National Science Foundation under award number IRI-9307895, and by grant No. 92-00163 from the United States - Israel Binational Science Foundation (BSF), Jerusalem, Israel. T.R. acknowledges support by the Academy of Finland under grant No. 18587.

Compressing the concordance serves several purposes. Not only does it save space, but it also saves processing time by reducing the number of I/O operations needed to fetch parts of the concordance into main memory (see [16, 6, 25]). Thus, to distribute a functional, full-text IRS, consideration must be given how to store the concordance efficiently. But concordance compression has theoretical interest as well. Current approaches to data compression tend to take a two stage approach: first one models the source, defining the message set and the probability of each message; then one creates the code for each message [2]. The concordance of a full text information retrieval system is the ideal entity on which to test this approach. As a well structured component of the IR system, it would seem particularly susceptible to modeling.

When choosing an appropriate compression scheme, we note that in a static IRS, compression and decompression are not symmetrical tasks. Compression is done only once, while building the system, whereas decompression is needed during the processing of every query and directly affects the response time. One may thus use extensive and costly preprocessing for compression, provided reasonably fast decompression methods are possible. Moreover, in an IRS, while we compress full files (text, concordance, etc.), we decompress only (though possibly many) short pieces on demand; these may be accessed at random by means of pointers to their exact locations. This limits the value of adaptive methods based on tables that systematically change from the beginning to the end of the file.

This paper is based on the following conceptualization of a concordance: every occurrence of a word in the database can be uniquely characterized by a sequence of numbers that gives its exact position in the text. Typically, such a sequence would consist of the document number d, the paragraph number p (in the document), the sentence number s (in the paragraph) and the word number w (in the sentence). This defines a hierarchy of four levels: to identify the location of a word, we need give only the quadruple (d, p, s, w), containing the *coordinate* of the occurrence. The concordance contains, for every word of the dictionary, the lexicographically ordered list of all its coordinates in the text.

In our implementation, we translate this model to an equivalent one, which we describe for a simpler two level hierarchy. In this represention, we indicate 1) the index of the next document containing the word, 2) the number of times the word occurs in the document, followed by 3) the list of word indices of the various occurrences:

$$\begin{array}{ll}
\text{word}_1: & (d_1,\ m_1\ ;\ \ w_{1,1}, w_{1,2}, \ldots, w_{1,m_1}) \\
 & (d_2,\ m_2\ ;\ \ w_{2,1}, \ldots, w_{2,m_2}) \\
 & \cdots \\
 & (d_N,\ m_N\ ;\ \ w_{N,1}, \ldots, w_{N,m_N}) \\
\text{word}_2: & \cdots
\end{array}$$

where d_i is the document number, m_i the number of occurrences of the given word in the i-th document, and $w_{i,j}$ the index within d_i of the j-th occurrence of the word. To fully represent the concordance, we must model each of the components of the coordinate, and develop compression methods appropriate for each entity.

We see that the concordance can be a rather complicated data structure, especially if it permits hierarchical access to the database. But one or more components indicated above can usually be conceptualized as a *bitmap* (for example, the values indicating which documents a term appears in, or which paragraphs within a document contain a term), with auxiliary information indicating the number of terms in a unit of the hierarchy. Some of the global auxiliary information can be stored in the dictionary; other mechanisms may be needed for other.

In this paper we shall focus on compressing those components of the concordance that can be represented as bitmaps. Thus, when we refer to a concordance below, we are thinking of a set of bitmaps, each indicating, for example, the documents (or other textual units) in which a word occurs. Formally, each vector is the occurrence map of a given word W. Each bit position corresponds to one document, and the bit in position i is 1 if and only if the word W appears in document i, i.e., there is at least one coordinate of W which has i in its d-field. In the sequel, we shall use the languages of bits or documents interchangeably.

In order to encode the values of successive bits of the bitmap efficiently, we must obtain a good underlying model for the bit-generation process. A variety of methods have been used [7, 24, 18, 25] to compress concordances. The model of [3] assumed that all documents are approximately of the same size, that there is no between-document clustering of term occurrences, and that within a single document, words are independently distributed. On the basis of these assumptions, probability distributions were derived for each term, describing the behavior of the variables comprising coordinates in the concordance. The probabilities corresponding to the actual coordinate values were then transformed to bits by using arithmetic encoding, though Huffman or Shannon-Fano codes could have been used as well.

In this paper we generalize the independence models to incorporate a tendency of terms to cluster. This is a natural extension of the independence model, since textual data contains inherent dependencies. For example, if the documents of an Information Retrieval system are grouped by author or some other criterion, the d-values of many terms will tend to appear in clusters because of the specific style of the author, or because adjacent documents might treat similar subjects. Similarly, term-occurrences will cluster *within* documents, reflecting content variations over a document.

In the next part of this paper, we begin our study of various models of clustering by introducing a Hidden Markov model (HMM) to represent bitmap generation. For computational reasons, we then switch to traditional Markov models which approximate the HMM. A set of criteria is developed in Section 2.3 to constrain the allowable set of n-state models, and a full inventory is given for $n \leq 4$. Graph oriented operations among the various models are defined in Section 3., and are used to simplify the organization of the models. Section 4. concentrates on 4-state-models. One of the models is analyzed in detail in 4.3. Similar results can be derived for the other models, and are given without the proofs in 4.4.

Section 5. presents the details of the experimental evaluation. The encoding algorithm is formally stated in Section 5.1. The following sections suggest ways for parameter and performace estimation. Finally, the new methods were tested on the concordances of the English Bible and of two large full-text retrieval systems: the *Trésor de la Langue Française* (TLF) and the *Responsa Retrieval Project* (RRP). The results, including comparisons with other compression methods that appeared in the literature, are presented in Section 5.4.

2. Models of Clustering

The most common method for compressing a concordance is a form of run-length encoding, perhaps using a variable length encoding scheme for representing the run lengths [21, 26]. In effect this assumes a model of statistically independent term occurrence. But terms don't occur independently. They tend to cluster over authors and over time. Within documents, they cluster over text segments discussing the concept represented by the term. We thus expect a model recognizing term clustering to represent more accurately the occurrences of terms, and to offer more effective compression capability.

Should the tendency for clustering be pronounced, codes based on assumptions of term independence will produce poor compression. In this section we look at some Markov models of clustering that potentially can be used to improve compression. The simpler independence model examined in [3], which ignores clustering effects, can be considered a degenerate Markov model with one state.

We shall conceptualize our bitmap as being generated as follows. At any bitmap site we are in one of two states: a cluster state (C), or a between-cluster state (B). Initially we are in one of these states, as determined by an initial probability distribution. Subsequently, governed by probabilities associated with the state we are in, we generate a bitmap-value of zero or one; then, again governed by a probability distribution determined by the current state, we enter a new state as we move to the next bit-map site. This process is continued iteratively until the full bitmap has been generated. Such a model has been referred to as a *Hidden Markov Model* (HMM) in the literature, and will be defined in more detail below.

Unfortunately, while we believe that such a model is likely to be a reasonable representation for most bitmaps that describe concordances, it is analytically difficult to use. The problem is that, even if we know the initial state, we cannot know for certain which state we are in at any bitmap location; even the complete observable history of the generation process is inadequate for determining the state sequence. This makes parameter estimation difficult, and also complicates the formulae needed for compression and decompression.

However, in many situations, the one-bits are strongly associated with cluster states, and the zero-bits with the between-cluster states. Thus, while we never know for certain which state we are in, if we have just generated a sequence of ones, or of zeroes, it is very likely we are in a cluster state, or between-cluster state, respectively. It is mainly when we believe we are in a cluster (between-cluster) state and generate a zero (one) that there is genuine confusion.

For such situations, an approximation to the HMM seems reasonable. We shall introduce several traditional Markov models. These models will have a Cluster state and a Between-cluster state, reflecting the underlying hidden Markov process. But if we are in a cluster (between-cluster) state and generate a zero (one) then we enter one of the other states, which acts as a transition or uncertainty state. In this way we can introduce some of the aspects of the HMM while enjoying the simplicity of traditional Markov analysis.

2.1 Hidden Markov Model

In a conventional Markov model, given the initial state, we always know the state we are in. The system moves from one state to another governed by the model's matrix of transition probabilies. In making a transition, it emits a symbol, and the symbol uniquely determines the next state.

In a HMM, if we are in a given state, there is a probability distribution describing the symbol that is emitted, and a different probability distribution determining the next state. Thus the probability of emitting a symbol has been separated from the probability of making a specific transition to the next state.

To define the HMM formally, we need three probability distributions: one giving the probability of starting in state i (distribution $\pi(i)$); one governing transitions between states i and j (distribution $A(i,j)$); and for each state i, a distribution which gives the probability of generating a given symbol k (distribution $B(i,k)$). For our models, comprised of two states and two characters, five parameters are needed to define the model completely. However, since the impact of the parameter determining the initial state-probability is quickly attenuated, we shall refer to the HMM as a four-parameter model, allowing us to concentrate on the state transitions and character emissions.

HMM's have been used widely in speech recognition [22] and recently also in the prediction of protein structure [1, 15] and information retrieval research [20]. In our application the model has two states, one for being in a cluster and another one for being between clusters. In Figure 1 we have an example of an HMM, taken from TLF: it corresponds to the word `extravagant`, which appears in 392 of the 39000 documents. The probability of starting in state S_1, $\pi(1)$, is 0.65 and of starting in state S_2 is 0.35. If we are in state S_1, we will stay there with probability $A(1,1) = 0.77$. In state S_1 the probability of emitting a zero bit is $B(1,0) = 1.00$. The probability of moving from state S_1 to state S_2 is small ($A(1,2) = 0.23$), but once we do move, the probability of emitting a 1-bit increases from 0 to 0.03; once in S_2, there is a high probability (0.72) that we return back to state S_1. Thus, the state S_1 corresponds to the between-cluster state and the state S_2 to the cluster state; in this special case of low 1-bit density, the two states are quite similar in their likelihood of generating a 1-bit.

To use the HMM we must first estimate its parameters. Initially, we choose arbitrary probabilities for all parameters. We then use an iterative procedure that, in each stage, improves the values of the parameters. More about the procedure and the implementation issues can be found in [22]. Once estimated, we store the parameter values of the model (four probabilities in the above model). Given the parameters, we can compute the equilibrium state probabilities of the model. In the actual encoding, we begin with these equilibrium state probabilities and compute the probabilities of a one or zero; this is used to encode the actual value—for example, with arithmetic encoding. The current state probabilities, the actual bit-value, and the transition matrix allow us to compute the state probabilities for the next bit site. The procedure continues in this manner until the entire bitmap is encoded.

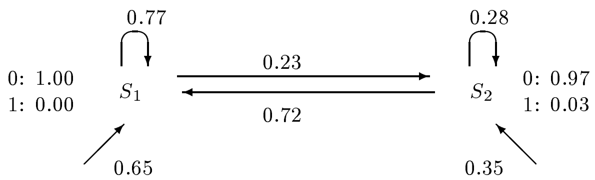


Figure 1: *Example of the simple HMM for* `extravagant`

2.2 Traditional Markov Model

We are conceptualizing bit-generation as a process in which we are in either a cluster or non-cluster state, and which state we are in determines the probabities of a 1/0 bit. An advantage of the HMM is that it recognizes the possibility that we can generate zeroes while we are in a cluster state, and ones in a non-cluster state.

The simplest traditional Markov model permitting clustering would likewise be comprised of two states: one representing the high probability of generating a 1 within a cluster, the other the between cluster condition. However, in such a traditional Markov model, the bit generated would determine the next state; thus, for example, a zero bit would cause us to leave the cluster state.

To incorporate the flexibility of the HMM within a traditional Markov framework, we introduce additional states. In this paper, we investigate a family of n-state Markov chains as models of how our bitmap was generated: we assume we know the initial state of the process; then, as we traverse the bitmap from beginning to end, at each position we are in a state, and that state determines the probability of being in any given state at the next location. Each such state transition is associated with a one or zero bit being generated. When we encode a bitmap, we simulate this process. We begin in the known initial state, and the bit-values scanned determines the state we are in and the probability of the next symbol we will scan; these probabilities are used to construct the code.

2.3 Hierarchy of Models

The simplest traditional Markov model is the degenerate model with a single state, equivalent to the independence model. Such a model cannot exhibit true clustering.

The simplest Markov model permitting clustering is the two state model referred to above. It is based on two states C and B, with the probability of a transition to C (that is, the probability of generating a 1-bit) depending on the state we are in. The state C indicates that we are in a cluster, and thus are more likely to generate occurrences of the designated term. We enter B as soon as we leave the cluster (that is, generate a zero-bit).

A limitation of the traditional two-state model is that it doesn't recognize the possibility of spurious zeroes within a cluster or spurious ones between clusters. That is, we would like to incorporate the possibility, inherent in the HMM studied above, that we may be in a cluster and generate zero bits without leaving the cluster, and conversely, being between clusters and generating 1-bits. We accommodate these possibilities by introducing transitional states.

A general n-state model has states C and B as above and $n-2$ transitional states $(X_1, X_2, \ldots, X_{n-2})$. It can be represented by a directed graph $G = (V, E)$ in the following way:

MM1. The set of vertices is $V = \{v_0, v_1, \ldots, v_{n-1}\} = \{C, X_1, \ldots, X_{n-2}, B\}$;

MM2. exactly two directed edges leave each vertex; one edge is labeled by 0, the other by 1, corresponding to the bit value generated in the transition represented by the nodes connected by the edge;

MM3. one node is designated as the start node; below we assume the system starts in state B;

MM4. transition probabilities (going from state i to state j) are assigned to the corresponding edges.

Thus, our graph resembles the transition diagram of a finite state automaton, with the additional assignment of transition probabilities. Such models allow us to simulate remaining in a cluster (or between clusters) even after generating a 0 (1), thereby preserving a very important property of the HMM. In this sense, we shall think of the transitional states as being accomplices of the C and B states: if generating a 0 in a C state takes us to a transitional state, this suggests that we are only tentively leaving the C state.

At first sight, an analysis of the graph associated with the Markov model suggests there are n^{2n} possible structures that can represent Markov models with n states: two edges leave each node, each of which can enter n possible nodes, so there are n^2 possibilities for the outgoing edges of each node; since there are n nodes, the total number of possible graphs in $(n^2)^n$. This means that, e.g., even for $n = 4$, the number is 65536, and for $n = 5$ it is almost 10 million. However, almost all of these are unreasonable as represention of our problem. We thus impose several additional conditions for a graph to be an acceptable representation of our Markov model. In summary, a *legitimate* Markov model can be represented by an edge-labeled graph, satisfying the following conditions (C1)–(C5):

(C1) Each node has exactly two edges leaving it, one labeled by a 1, the other by a 0.

(C2) There are no multiple edges: there are no states I and J such that the system can go from I to J by either edge, emitting either a one or a zero. Otherwise, we would have states that served only as lag or delay states, with the symbol emitted when within such a state not providing real information; there is no physical justification within our problem for introducing lag states.

(C3) The graph is strongly connected: each node can be reached from any other node.

(C4) There is a loop on C and one on B, but the intermediate states X_i cannot have loops. Thus the transitional states are indeed transitional: we can't remain in any transitional state while generating runs of 1's or 0's; the former should lead to state C, the latter to state B. And, once we are in C (B), we remain there as long as we observe 1's (0's).

(C5) All edges entering C are labeled with a 1; all edges entering B are labeled with a 0. This is true in particular for the loops on C and B.

One can represent each graph G by an incidence matrix $M_G = (c_{ij})$. Because of conditions (C1) and (C2), it seems natural to set c_{ij} as the label of the unique edge from v_i to v_j ($0 \leq i, j < n$) when such an edge exists; however, we want to reserve the zero value to denote the absence of an arc. We thus use the following consistent labelling:

$$c_{ij} = \begin{cases} 0 & \text{if } (v_i, v_j) \notin E \\ 1 & \text{if } v_i \xrightarrow{1} v_j \\ 2 & \text{if } v_i \xrightarrow{0} v_j \end{cases}$$

where, here and below, we use the notation $v_1 \xrightarrow{\alpha} v_2$ if there is a directed edge from node v_1 to node v_2 and it is labeled $\alpha \in \{0, 1\}$.

Thus, because of (C1), each row in M_G contains a 1, a 2 and $n-2$ zeros. For example, the incidence matrix corresponding to the graph labeled 4S1 in Figure 2 is $\begin{pmatrix} 1\,2\,0\,0 \\ 0\,0\,1\,2 \\ 1\,2\,0\,0 \\ 0\,0\,1\,2 \end{pmatrix}$, and that of 4C1 is $\begin{pmatrix} 1\,2\,0\,0 \\ 1\,0\,2\,0 \\ 1\,0\,0\,2 \\ 1\,0\,0\,2 \end{pmatrix}$.

Condition (C5) implies that in the first (last) column, corresponding to C (B), there are only 1's (2's) or 0's. From (C4) and (C5) follows that $c_{00} = 1$, $c_{n-1,n-1} = 2$ and $c_{ii} = 0$ for $0 < i < n-1$. To check the strong connectivity (C3), note that $(M_G)^k$ contains a non-zero value in its (i, j) position, if and only if there is a directed path from v_i to v_j in G of length exactly k edges. Thus G is strongly connected if and only if there are no null entries in the matrix $\sum_{k=1}^{n-1}(M_G)^k$

The above conditions impose severe constraints on the incidence matrix and permit an explicit enumeration of the legitimate graphs. Further, many are relabeling of others, with X_i and X_j interchanged, for certain i and j. This is easily tested: if by interchanging the X_i and X_j rows and columns of a matrix gives another legitimate matrix, one of the two models can without loss be eliminated. For the case $n = 4$, this reduces the number of models from 65536 to 21. We can remove an additional 8 models if we introduce, as well, the following plausible condition, which supports our interpretation of the states X_i as transitional states.

(C6) If k zeros (ones) are needed to go from C to B (from B to C), and there is a path of length ℓ from X_i to B (C) all of whose edges are labeled by zeros (ones), then $\ell \leq k$. For example, we considered a graph to be unreasonable if a zero value takes us directly from C to B, but it requires two zeros to go from a transition state to B.

3. Structure of Models

The models described above have a structure that is best described in terms of two general sets of operations: *Complementation* and *Reduction*. These will be described in the following sections.

3.1 Complementation Operator

The set of resulting graphs has an interesting internal structure: It is possible to divide these graphs into two classes if we introduce the concept of Complementation ($\mathcal{C}$) functions, and thereby, of a $\mathcal{C}$-symmetric graph.

In general, given an edge-labeled graph, G, we define its complementary graph, $\mathcal{C}(G)$ by means of a pair of self inverting complementation functions $\mathcal{C} = (\mathcal{C}_S, \mathcal{C}_L)$, defined respectively on the nodes and on the labels of G.

- $\mathcal{C}(G)$ has the same set of nodes and labels as G. Since $\mathcal{C}$ is self-inverting, any node of $\mathcal{C}(G)$ is the image of some v in G, and similarly for the labels.

- $\mathcal{C}(G)$ has an edge, labeled by $\mathcal{C}_L(\alpha)$, between nodes $\mathcal{C}_S(v_i)$ and $\mathcal{C}_S(v_j)$, for $v_i, v_j \in V$, if and only if G has an edge labeled α from v_i to v_j.

Thus, if we are given a path $S_1 S_2 \cdots S_r$ between nodes S_1 and S_r of G, associated with the sequence of labels $\alpha_1 \alpha_2 \cdots \alpha_{r-1}$, then a *$\mathcal{C}$-complementary* path $\mathcal{C}_S(S_1)\mathcal{C}_S(S_2)\cdots\mathcal{C}_S(S_r)$, with labels $\mathcal{C}_L(\alpha_1)\mathcal{C}_L(\alpha_2)\cdots\mathcal{C}_L(\alpha_{r-1})$, exists in $\mathcal{C}(G)$ between nodes $\mathcal{C}_S(S_1)$ and $\mathcal{C}_S(S_r)$. For the models we are considering, we insist in addition that $\mathcal{C}_S(B) = C$ and that $\mathcal{C}_L(1) = 0$. With these conditions, we can easily see that a graph G satisfies conditions (C1) to (C6) if and only if the $\mathcal{C}$-complementary graph $\mathcal{C}(G)$ does. Thus the complementation operator defines a structure of our set of models as follows.

A labeled graph is *$\mathcal{C}$-symmetric* if $\mathcal{C}(G) = G$. Thus, for every path in G, the corresponding $\mathcal{C}$-complementary path also exists G. This condition is equivalent to being able to order the nodes so that complementary elements are at positions which are symmetric relative to the middle point of the matrix, i.e., c_{ij} and $c_{n-1-i,n-1-j}$ are either both zero, or, if the one is 1, the other is 2, for $0 \le i, j < n$. Thus the first matrix mentioned above, corresponding to 4S1, satisfies the symmetry condition, but the second matrix does not. (A similar observation is valid for more general graphs. If the entries in the incidence matrix are the actual labels of arcs, or the null value (<*null*>) for nodes that aren't connected, then $c_{ij} = \mathcal{C}_L(c_{n-1-i,n-1-j})$ if $c_{i,j}$ isn't null, and $c_{ij} =$<*null*> otherwise, for $0 \le i, j < n$. The value <*null*> could be any number distinct from the arc labels.)

We shall only consider cases where $\mathcal{C}_L$ is binary complementation, though the concept can be generalized to any self-inverting function $\mathcal{C}_L$. In terms of our complementation functions, we define $\mathcal{C}_S(B) = C$, and $\mathcal{C}_S(X_1) = X_2$; then the thirteen acceptable models divide into three symmetric models and five pairs of models, each pair of which is comprised of a model and its complement.

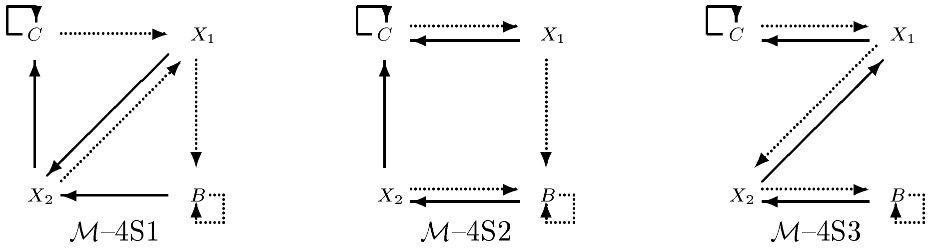
While particularly useful for organizing our models, the complementation operator is generally applicable to edge-labeled graphs. For example, the well-known *de Bruijn* diagrams are $\mathcal{C}$-symmetric. In fact, the model labeled 4S1 in Figure 2 is the *de Bruijn* diagram $G_{2,3}$ (see [11, Section 4.3]).

The 13 graphs of the 4-state models satisfying the conditions (C1)–(C6), grouped by complementation relation, are schematically displayed in Figure 2, and the graphs for all the models with $n \le 3$ are displayed in Figure 3. The labeling is implicit in the form of the arrows: a solid line representing an edge labeled 1, and a dotted line an edge labeled zero. For ease of description, we shall refer in the sequel to the models by their labels in the figures, which mention the number of their states; e.g., $\mathcal{M}$–4S1 will refer to the model in the upper leftmost corner of Figure 2.

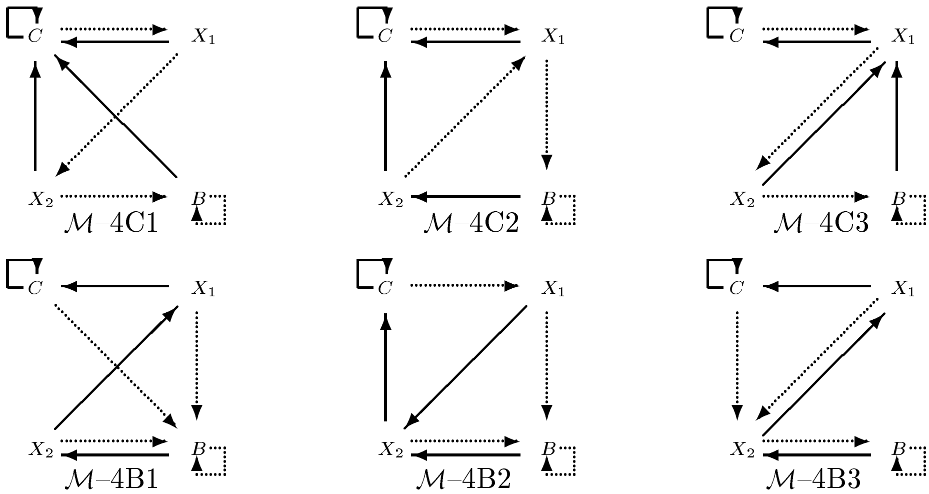
The following conventions were used for naming the various models: each model name consists of a pair nL or a triplet nLi, where n is the number of states, $L \in \{S, C, B\}$ indicating whether the model is $\mathcal{C}$-symmetric, i.e., states B and C play symmetric roles, or whether the model is biased towards the C state or the B state, respectively; i is a running index distinguishing graphs that would otherwise share a given name. We say that a model is C-biased, if it is harder to get from C to B than vice versa; that is, one needs more zeros to get from C to B than one needs ones for the opposite path. For $\mathcal{M}$–4C2 and $\mathcal{M}$–4B2 these paths are both of length 2; we consider $\mathcal{M}$–4C2 to be C-biased because starting in C and generating the sequence 01, we get back to C, whereas starting in B and generating 10 gets us to an intermediate state. In other words, a C-biased model, once in state C, is more reluctant to accept a 0 as an indicator of having left the cluster than would be the case in the complementary situation.

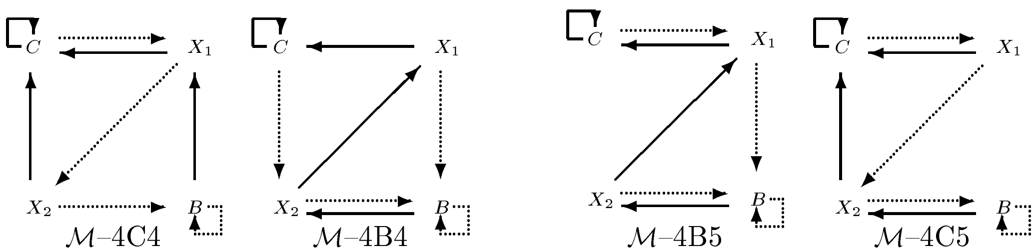
3.2 Reduction Operator

We shall demonstate below how various properties of our models can be derived, given the model parameters. However, we can save much effort by noting a special relation that exists between some lower and higher order models: this relationship allows us to directly derive the results for the lower order model given the corresponding results for the higher order model. In this section we shall



$\mathcal{C}$-symmetric 4-state models





Asymmetric 4-state models, grouped in pairs of mutually $\mathcal{C}$-symmetric models

Figure 2: *Inventory of 4-state Markov models*

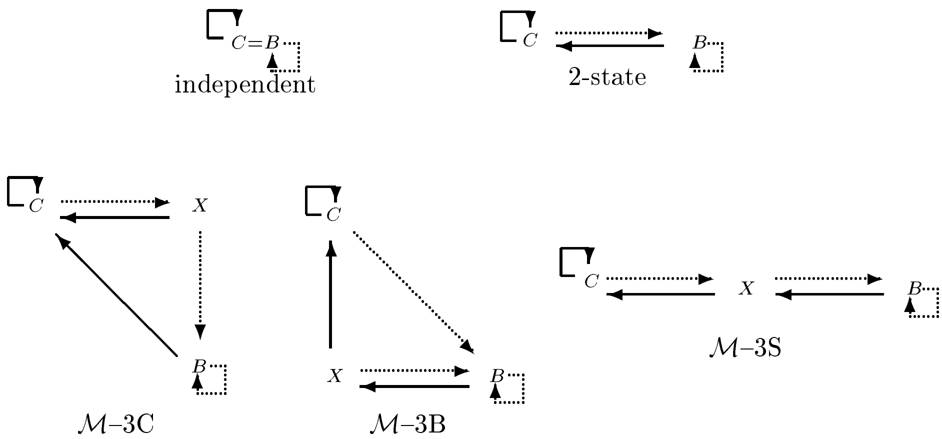


Figure 3: *1-, 2- and 3-state Markov models*

explore this relation, which allows us to consider certain models as generalizations of lower order models. To do this we define the notion of *reducibility* among models.

We first review the formal structure of our Markov models. Each consists, subject to restrictions as stated above, of:

- A graph;
- Labels for edges;
- An initial probability for each node (here, the probability of B when starting is 1); and,
- Transition probabilities.

Suppose we are given such a structure. We wish to determine when an arbitrary partition of the nodes defines a graph that itself represents a legitimate Markov model of the type we are considering; such partitions will be called *reductions*, an idea closely related to *lumpability*, as defined, for example, in ref. [5].

Consider a graph, representing a Markov model, with node set $V = \{v_0, \ldots, v_{n-1}\}$. Let $\mathcal{P} = \{V_1, \cdots, V_n\}$ be a partition of V. If nodes v_1 and v_2 of V are in the same class V_i, we write $v_1 \equiv v_2$. Such a partition is called a *reduction* if:

a) $\mathcal{P} = \{V\}$,

which is the degenerate case for which the partition consists of a single element — that is, all the states are merged into a single one, and the graph consists of a single vertex; or,

b) if the partition contains more than one element and satisfies the following conditions on the underlying graph structure:

(R1) Nodes C and B are in different members of the partition.

(R2) Whenever an arc labeled with α goes from nodes v_1 to w_1, then, for any $v_2 \equiv v_1$, if an arc labeled with α connects v_2 with w_2, then $w_2 \equiv w_1$. That is, all arcs leaving a class and having the same label are directed to the same target class.

More succinctly, we have:

$$\big((v_1 \xrightarrow{\alpha} w_1) \quad \wedge \quad (v_1 \equiv v_2) \quad \wedge \quad (v_2 \xrightarrow{\alpha} w_2)\big) \qquad \Longrightarrow \qquad (w_1 \equiv w_2),$$

which is schematically represented in Figure 4(a).

(R3) Whenever an arc labeled with α goes from nodes v_1 to w_1, then, for any $v_2 \equiv v_1$, if an arc labeled with β, with $\beta \neq \alpha$, connects v_2 with w_2, then $w_2 \not\equiv w_1$. That is, if $\alpha \neq \beta$:

$$\big((v_1 \xrightarrow{\alpha} w_1) \quad \wedge \quad (v_1 \equiv v_2) \quad \wedge \quad (v_2 \xrightarrow{\beta} w_2)\big) \qquad \Longrightarrow \qquad (w_1 \not\equiv w_2),$$

as in Figure 4(b). Note that for the binary case we consider, we have $\beta = 1 - \alpha$. In other words, for two given classes V_i and V_j of the partition, all the edges connecting any node in V_i to any node in V_j must be labeled identically.

R2 and R3 tell us that all arcs between two classes have the same label, and that all arcs leaving a class with the same label go to the same target class. For example, for model $\mathcal{M}$–4S2, one could define $V_1 = \{C, X_2\}$ and $V_2 = \{B, X_1\}$; then all the edges going from V_1 to V_2 are labeled 0, and all those going from V_2 to V_1 are labeled 1.

(R4) If $v \equiv C$, then all edges terminating on v must be labeled by 1; similarly, if $v \equiv B$, then all edges terminating on v must be labeled by 0.

(R5) There can be no internal edges in any component not containing either C or B.

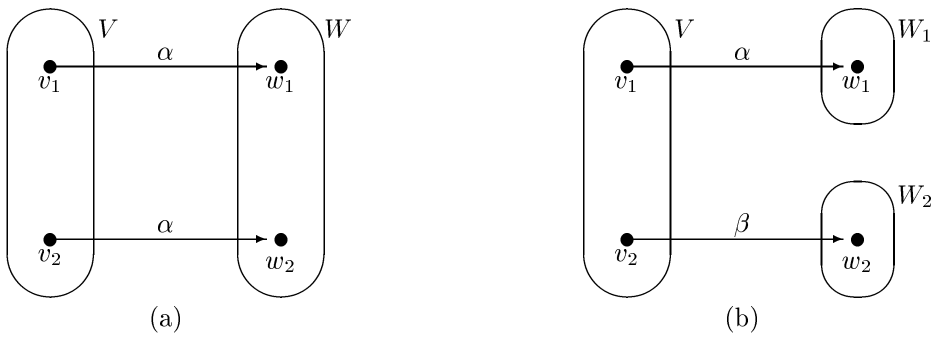


Figure 4: *Schematic representation of reduction partitions*

In addition, to be consistent with the Markov property, we require that:

(R6) If $v_i \equiv v_j$, then the probability of a transition to a node in V_k is the same for v_i and v_j. Specifically, v_i and v_j have the same probability of generating a 1 or a 0.

If we are given a graph G and a reduction, then we can define a *reduced* Markov graph G' as follows:

(RG1) Nodes: $\mathcal{P}$.

(RG2) Edges: If $v_1 \xrightarrow{\alpha} v_2$ in G, with $v_1 \in V_1$ and $v_2 \in V_2$, then $V_1 \xrightarrow{\alpha} V_2$ in G'. This defines a single edge labeled α leaving any node of G', because of condition (R2), and the definition is independent of the specific nodes selected because of condition (R3). If $V_1 = V_2$, then the edge is a loop, which is possible only for a node containing C or B.

(RG3) Initial probabilities: In general, for a node V_i of G', the initial probability is just the sum of the initial probabilities in G of the constituent nodes of V_i. For the case at hand, this will be zero for all the nodes, except the one containing B, for which it is 1.

(RG4) Transition probabilities: if V_1 and V_2 are nodes in G', we define the transition probability between V_1 and V_2 as

$$T_{V_1,V_2} = T_{v_i,v_j}\,,$$

for $v_i \in V_1, v_j \in V_2$, and T_{v_i,v_j} the transition probability between v_i and v_j in G. (Note that by (R6), if $V_1 \xrightarrow{\alpha} V_2$, then $v_i \xrightarrow{\alpha} v_j$ for all $v_i \in V_1$ and $v_j \in V_2$). Below, when we derive properties of lower order from higher order models, we will implicitly equate the above T_{v_i,v_j} and set the transition probabilities between v_i and v_j to the common value.

(RG5) Special vertices C and B: The component in G' containing C of G is denoted C, and similarly for the component containing B. (For completeness, if there is a single component, we call the node C.)

The reduction relation is very much like a congruence in algebra, with the reduced graph like a factor-structure.

Several consequences follow from this definition, taken in conjunction with the restrictions defining a legitimate graph structure. The most important is that the reduced graph has all the properties we demand of a legitimate graph.

We assume below that we have at least two components.

(C'1) Each node in G' has exactly two edges leaving it, one labeled by 1, the other by 0; this is a consequence of (RG2) given the defining property (C1) of G.

(C'2) There are no multiple edges, because of (R3).

(C'3) G' is strongly connected, because G is.

(C'4) In G', as in G, there will be a 1-loop attached to C and a 0-loop attached to B. Further, all edges labeled 1 in the C component of G' must be internal to the C component, because of (R2) and the requirement that C in G has a loop; and all edges labeled by 0 and starting in the C component must leave the component, because of condition (R3): if an arc labeled with 0 connected any vertex in C with a different vertex in C, then all 0-labeled arcs starting in C must end in C; since this is true as well for 1-labeled arcs, no path starting with C could connect to B, contradicting the assumption that G is strongly connected. Similar conditions, for edges labeled 0 and 1 respectively, are true for the B component. Transitional states can have no loops, because of (R5).

(C'5) All edges entering C are labeled by 1 and all edges entering B are labeled by 0, by (R4).

Looking forward, the notion of a reduction will allow us to deduce properties of lower order models, given properties of higher order models. Intuitively, if some analysis assigns a probability to each node in G, the parallel probability for a node in the reduced graph should be the sum of the corresponding probabilities of the original graph — this is true, for example, for equilibrium probabilities.

Similarly, both the original and reduced structures define a probability for any sequence of 1's and 0's. We will be interested in properties that depend on probabilities of strings for which we can deduce the value for the reduced graph by equating the transistion probabilities in the corresponding value for the initial graph. This is the case for the expected number of clusters.

Figure 5 displays all the possible reductions among the n-state models we consider, for $n \leq 4$. Models that are $\mathcal{C}$-symmetric are indicated by a double borderline. If $\mathcal{M}_1$ can be reduced to $\mathcal{M}_2$,

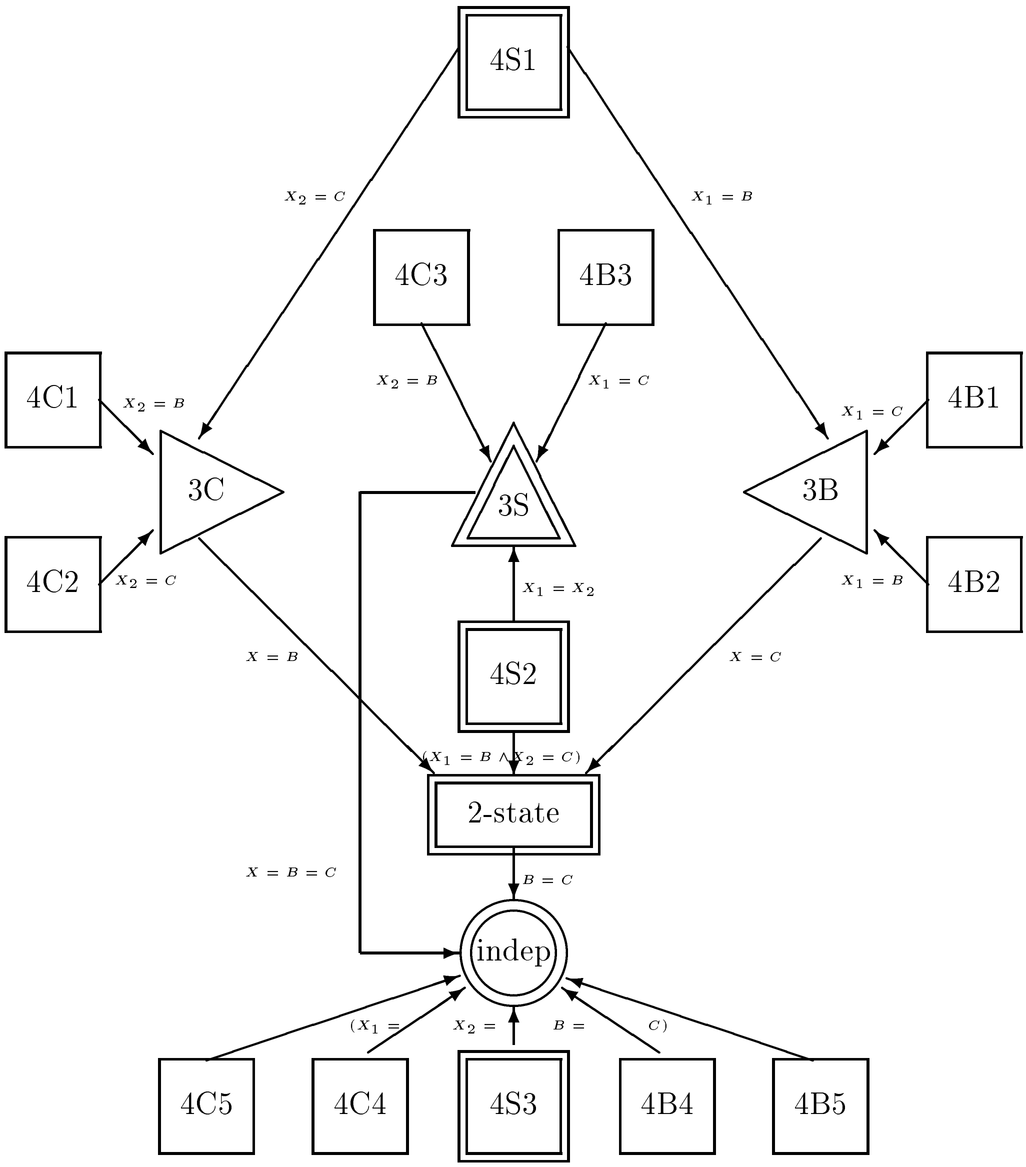


Figure 5: *Reductions among the various Markov models*

this is indicated by an arrow from $\mathcal{M}_1$ to $\mathcal{M}_2$, near to which appear the states to be put in the same class of the partition.

We note that $\mathcal{M}$–4S1 reduces to the 3-state model $\mathcal{M}$–3C by identifying state[1] X_2 with state C, and to $\mathcal{M}$–3B by identifying state X_1 with state B. $\mathcal{M}$–4S2 reduces to $\mathcal{M}$–3S by identifying the two transitional states X_1 and X_2. Thus, all the 3-state models (as well as the 2- and 1-state models) previously studied are special cases of the symmetric 4-state models.

Note also that $\mathcal{M}$–4S2 does not reduce to another 3-state model beside $\mathcal{M}$–3S; only by identifying both X_1 with B and X_2 with C do we get an acceptable partition, that corresponding to the 2-state model. For $\mathcal{M}$–4S3 and the other models appearing at the bottom of Figure 5, an acceptable partition is obtained only when all four states are merged, yielding the independent 1-state model.

We finally draw attention to the symmetric aspect of the reduction diagram, which reflects the fact that for each reduction from one model to another, there is a corresponding reduction between $\mathcal{C}$-symmetric counterparts.

Further consequences include:

- If there is an edge labeled α emanating from a node in a given class V_i, and incident onto a node outside of V_i, then there cannot be an edge labeled α from one node to another in V_i. That is, if an edge labeled α connects v_1 and v_2, for $v_1 \equiv v_2$, then all edges labeled α starting in the equivalence class defined by v_1 and v_2 must terminate in that class. This is in particular true for edges labeled by 1(0) in the class containing C (B). For example, again in $\mathcal{M}$–4S2, the set $V' = \{C, X_1\}$ cannot be a class in a partition, because there is an edge labeled 0 leaving V', but $C \xrightarrow{0} X_1$. This property is a consequence of (R2).
- If from a given vertex v, there is an edge labeled 0 to w_1, and another edge labeled 1 to w_2, then w_1 and w_2 cannot possibly belong to the same class in a partition, unless v too belongs to that class. For example, in $\mathcal{M}$–4S1, we have $X_1 \xrightarrow{0} B$ and $X_1 \xrightarrow{1} X_2$, implying that the set $\{X_2, B\}$ may not be part of a partition. This is a consequence of (R3).
- To any path in a reduced graph, G', there corresponds at least one path in the original graph G. Denote a path in G' by the sequence *node (output symbol) node ... node*. Then, given any such path $P = V_0(\alpha_0)V_1(\alpha_1)V_2(\alpha_2)...V_n$, and starting with *any* node of G included in V_0, the path in G associated with $\alpha_0\alpha_1 \cdots \alpha_{n-1}$ is consistent with P. This is the essential idea behind the graph congruence and is why everything works.

4. 4-state Markov models

There is a natural tradeoff when deciding the order n of the Markov model to be used. The higher n, the better can the model describe the bit generation process — but more bits are needed to communicate the model parameters to the decoder. The independence model of [3] corresponds to $n = 1$ and in [4], we considered $n = 3$. To maintain the same complexity as the 2-state HMM of Section 2.1 (which has four parameters), in this section we shall focus on 4-state Markov models.

4.1 Interpretation of the models

The models we have developed are intended to be simplifications of the HMM. To make explicit the relationship between these two types of models, we describe one in detail.

The scenario represented by $\mathcal{M}$–4S2, for example, is as follows. Suppose we start within a cluster, that is in state C. As long as ones are generated, we remain within the cluster. However, if

[1] More precisely, identifying the probability of generating a 1 in state X_2 with that of state C.

a zero is generated, it suggests the possibility of having left the cluster, but the possibility remains that we are still in C, and that the 0 is just a random occurrence. This possibility, consistent with the underlying HMM, is taken into account by entering the transitional state X_1. If, once in X_1, the next bit generated is a 1, we go back to the cluster state and conclude that the preceding zero was merely an accident; if, however, a second zero is produced, we conclude that we have left the cluster and are now in state B. State B remains then in effect as long as zeros are generated, and the behavior is symmetrical. That is, if a 1 is generated, we are not yet sure it is the beginning of a cluster, so we go into a transitional state X_2. A second 1 confirms that we are in a cluster, so we go to C, but a zero drops us back to B.

Below, we shall analyze in detail model $\mathcal{M}$–4S1, and indicate without proof the results for the other $\mathcal{C}$–symmetric models. In addition, for illustration and contrast, we shall present the results for one of the asymmetric models that performed well in our experiments, as well as of its $\mathcal{C}$–symmetric counterpart.

4.2 Relation to earlier models

In earlier work [3], we assumed an independence model, which is equivalent to a single state Markov model that ignores that word occurrences may cluster. A primitive clustering effect can be described by a two-state Markov model, with only a Cluster-State and a Between-Cluster state. But such a model has no tolerance for zeros (one) in a Cluster (Between-Cluster) state.

The model discussed in [4] is $\mathcal{M}$–3C. It differs from $\mathcal{M}$–4S1 in that a single 1 is enough to make the transition from B to C. In the complementary three-state model $\mathcal{M}$–3B, the transitional state is entered on the way from B to C: a single zero is enough to leave the cluster, but at least two consecutive 1's are needed to pass from the between to the cluster state. The third possible 3-state model $\mathcal{M}$–3S is $\mathcal{C}$-symmetric, e.g., two zeros are needed to get from C to B, and two ones are needed to pass from B to C. These three models exhaust this set of 3-state models satisfying the plausability requirements enumerated above.

The reason that $\mathcal{M}$–3B, the complementary model of $\mathcal{M}$–3C, has not been mentioned in [4] is because of the interpretation of the bit-vectors in our application. The ones correspond to documents containing the designated term, the zeros to documents in which the term does not occur. Now most of the terms of any natural language database occur in only a small part of the documents, so that the probability of a 1-bit is much smaller than that of a 0-bit. It is thus reasonable to allow single zeros to appear within a cluster as in $\mathcal{M}$–3C, while the occurrence of a single 1, which in itself is a relatively rare event, can already be considered as indicating the start of a new cluster. It might thus well be that for our applications the symmetry condition should not hold, and indeed, some non-symmetric 4-state models peformed consistently better in our experiments, reported in Section 5..

Our goal for the 4-state models is to predict the size of the gaps separating documents that contain the term and then deduce corresponding quantities for the 3-state models, 2-state models and the independence model. Gap length is an important characteristic of clustering, and the distribution of gap lengths distinguish a clustered model from the independence models. We shall derive formulae that indicate how the gap lengths depend on the underlying parameters for the various models. These formulae also offer an indication of whether clustering is a factor for a particular set of bitmaps.

4.3 Analysis of $\mathcal{M}$–4S1

Let us define the transition probabilities for $\mathcal{M}$–4S1 by:

$$
\begin{array}{rcl rcl}
C & \longrightarrow & C : \Gamma_C, & C & \longrightarrow & X_1 : 1 - \Gamma_C; \\
X_1 & \longrightarrow & X_2 : \Gamma_{X_1}, & X_1 & \longrightarrow & B : 1 - \Gamma_{X_1}; \\
X_2 & \longrightarrow & C : \Gamma_{X_2}, & X_2 & \longrightarrow & X_1 : 1 - \Gamma_{X_2}; \\
B & \longrightarrow & X_2 : \Gamma_B, & B & \longrightarrow & B : 1 - \Gamma_B.
\end{array}
$$

That is, if we are in state S, Γ_S is the probability of emitting a 1.

We next compute several interesting properties of this model.

4.3.1 Equilibrium Probabilities

We first compute the long run equilibrium probabilities for these states. Let $\pi = [\pi_C, \pi_{X_1}, \pi_{X_2}, \pi_B]$ be the vector of probabilities of being in states C, X_1, X_2, B respectively in the long run. Ordering the states as induced by π, we get the transition matrix

$$
T = \begin{pmatrix} \Gamma_C & 1-\Gamma_C & 0 & 0 \\ 0 & 0 & \Gamma_{X_1} & 1-\Gamma_{X_1} \\ \Gamma_{X_2} & 1-\Gamma_{X_2} & 0 & 0 \\ 0 & 0 & \Gamma_B & 1-\Gamma_B \end{pmatrix}.
$$

Since π satisfies $\pi = \pi T$, we have the following system of equations

$$
\begin{aligned}
\pi_C &= \pi_C \Gamma_C + \pi_{X_2} \Gamma_{X_2} && (1) \\
\pi_{X_1} &= \pi_C (1-\Gamma_C) + \pi_{X_2}(1-\Gamma_{X_2}) && (2) \\
\pi_{X_2} &= \pi_{X_1} \Gamma_{X_1} + \pi_B \Gamma_B && (3) \\
\pi_B &= \pi_{X_1}(1-\Gamma_{X_1}) + \pi_B(1-\Gamma_B). && (4)
\end{aligned}
$$

In addition, we also have the condition $\pi_C + \pi_{X_1} + \pi_{X_2} + \pi_B = 1$. Now solving the equations for π_C, π_{X_1}, π_{X_2} and π_B, we get

$$
\begin{aligned}
\pi_C &= \frac{\Gamma_{X_2}\Gamma_B}{\Gamma_{X_2}\Gamma_B + 2(1-\Gamma_C)\Gamma_B + (1-\Gamma_C)(1-\Gamma_{X_1})} && (5) \\
\pi_{X_1} = \pi_{X_2} &= \frac{(1-\Gamma_C)\Gamma_B}{\Gamma_{X_2}\Gamma_B + 2(1-\Gamma_C)\Gamma_B + (1-\Gamma_C)(1-\Gamma_{X_1})} && (6) \\
\pi_B &= \frac{(1-\Gamma_C)(1-\Gamma_{X_1})}{\Gamma_{X_2}\Gamma_B + 2(1-\Gamma_C)\Gamma_B + (1-\Gamma_C)(1-\Gamma_{X_1})}. && (7)
\end{aligned}
$$

4.3.2 Run Lengths

Each bit in a bitmap can be individually encoded, using arithmetic coding based on the probability of a one-bit at each stage; this is the encoding algorithm used in our experiments and described in more detail in Section 5.. Alternatively, it may be more convenient to encode the *gaps* between one bits. The reason why this alternative representation is equivalent is shown in the Appendix. To encode the gap sizes, we need their distribution, which is easily derived using the above model.

We thus compute the probability of a run of k zeroes, following a 1. Observe that if a 1 was just generated, the associated transition must have left us in state C or state X_2. Thus denoting by π'_C

and π'_{X_2} the probabilities of being in state C and X_2 respectively after we have just generated a 1, we have $\pi'_C + \pi'_{X_2} = 1$. Let $p_{k|C}$ be the probability of a run of k zeroes followed by a one, given that we are initially in state C. Then

$$\begin{aligned}
p_{0|C} &= \Gamma_C; \\
p_{1|C} &= (1-\Gamma_C)\Gamma_{X_1}; \\
&\vdots \\
p_{n|C} &= (1-\Gamma_C)(1-\Gamma_{X_1})(1-\Gamma_B)^{n-2}\Gamma_B, \qquad \text{for } n \geq 2.
\end{aligned}$$

We define $p_{k|X_2}$ similarly as $p_{k|C}$. Then

$$\begin{aligned}
p_{0|X_2} &= \Gamma_{X_2}; \\
p_{1|X_2} &= (1-\Gamma_{X_2})\Gamma_{X_1}; \\
&\vdots \\
p_{n|X_2} &= (1-\Gamma_{X_2})(1-\Gamma_{X_1})(1-\Gamma_B)^{n-2}\Gamma_B, \qquad \text{for } n \geq 2.
\end{aligned}$$

Let p_k be the probability of a run of exactly k zeroes following a 1. Then

$$p_k = p_{k|C}\pi'_C + p_{k|X_2}\pi'_{X_2},$$

so we have

$$\begin{aligned}
p_0 &= \Gamma_C\pi'_C + \Gamma_{X_2}\pi'_{X_2}; \\
p_1 &= (1-\Gamma_C)\Gamma_{X_1}\pi'_C + (1-\Gamma_{X_2})\Gamma_{X_1}\pi'_{X_2};
\end{aligned} \tag{8}$$

and for $n \geq 2$,

$$p_n = (1-\Gamma_{X_1})(1-\Gamma_B)^{n-2}\Gamma_B\left[(1-\Gamma_C)\pi'_C + (1-\Gamma_{X_2})\pi'_{X_2}\right]. \tag{9}$$

It remains to compute π'_C and π'_{X_2}. Observe that if we are in state C, then the previous state must have been C or X_2. Therefore

$$\pi'_C = \frac{\pi_C\Gamma_C + \pi_{X_2}\Gamma_{X_2}}{p(1)}$$

where $p(1)$ is the probability of generating a 1. But

$$p(1) = \pi_C\Gamma_C + \pi_{X_1}\Gamma_{X_1} + \pi_{X_2}\Gamma_{X_2} + \pi_B\Gamma_B.$$

Hence

$$\pi'_C = \frac{\pi_C\Gamma_C + \pi_{X_2}\Gamma_{X_2}}{\pi_C\Gamma_C + \pi_{X_1}\Gamma_{X_1} + \pi_{X_2}\Gamma_{X_2} + \pi_B\Gamma_B}. \tag{10}$$

Similarly, if we are in state X_2, then the previous state must have been X_1 or B, and

$$\pi'_{X_2} = \frac{\pi_{X_1}\Gamma_{X_1} + \pi_B\Gamma_B}{\pi_C\Gamma_C + \pi_{X_1}\Gamma_{X_1} + \pi_{X_2}\Gamma_{X_2} + \pi_B\Gamma_B}. \tag{11}$$

Substituting (1) and (3) from the equations for the steady state probabilities into (10) and (11), we find that

$$\pi'_C = \frac{\pi_C}{\pi_C + \pi_{X_2}}, \qquad \pi'_{X_2} = \frac{\pi_{X_2}}{\pi_C + \pi_{X_2}}. \tag{12}$$

Substitute the expressions for π_C and π_{X_2} as functions of the Γ's using (5) and (6), we get after some simplification

$$\pi'_C = \frac{\Gamma_{X_2}}{1-\Gamma_C+\Gamma_{X_2}}, \qquad \pi'_{X_2} = \frac{1-\Gamma_C}{1-\Gamma_C+\Gamma_{X_2}}. \tag{13}$$

4.3.3 Expected Run Length

Using the above expressions for p_n and π', we can calculate the average size of a gap as:

$$
\begin{aligned}
\sum_{n=0}^{\infty} np_n &= (1-\Gamma_C)\Gamma_{X_1}\pi'_C + (1-\Gamma_{X_2})\Gamma_{X_1}\pi'_{X_2} \\
&\quad + (1-\Gamma_{X_1})\left[(1-\Gamma_C)\pi'_C + (1-\Gamma_{X_2})\pi'_{X_2}\right]\Gamma_B \sum_{n=2}^{\infty} n(1-\Gamma_B)^{n-2} \\
&= (1-\Gamma_C)\Gamma_{X_1}\pi'_C + (1-\Gamma_{X_2})\Gamma_{X_1}\pi'_{X_2} \\
&\quad + (1-\Gamma_{X_1})\left[(1-\Gamma_C)\pi'_C + (1-\Gamma_{X_2})\pi'_{X_2}\right]\frac{1+\Gamma_B}{\Gamma_B} \\
&= \frac{(1-\Gamma_C)(1-\Gamma_{X_1}+\Gamma_B)}{\Gamma_B}\pi'_C + \frac{(1-\Gamma_{X_2})(1-\Gamma_{X_1}+\Gamma_B)}{\Gamma_B}\pi'_{X_2} \\
&= \frac{1-\Gamma_{X_1}+\Gamma_B}{\Gamma_B}\left[(1-\Gamma_C)\pi'_C + (1-\Gamma_{X_2})\pi'_{X_2}\right] \\
&= \frac{1-\Gamma_{X_1}+\Gamma_B}{\Gamma_B}\left[\frac{(1-\Gamma_C)\Gamma_{X_2}}{1-\Gamma_C+\Gamma_{X_2}} + \frac{(1-\Gamma_{X_2})(1-\Gamma_C)}{1-\Gamma_C+\Gamma_{X_2}}\right] \\
&= \frac{(1-\Gamma_{X_1}+\Gamma_B)(1-\Gamma_C)}{\Gamma_B(1-\Gamma_C+\Gamma_{X_2})} \\
&= (1-\Gamma_C)\left(1+\frac{1-\Gamma_{X_1}}{\Gamma_B}\right)\left(\frac{1}{1-\Gamma_C+\Gamma_{X_2}}\right). \qquad (14)
\end{aligned}
$$

In fact, we are only interested in genuine gaps, that is, gaps of length at least 1. The expected genuine gap size is $\frac{1}{1-p_0}\sum_{n=1}^{\infty} np_n$; but from (8) and (13), we get

$$1-p_0 = \frac{\Gamma_C\Gamma_{X_2}+\Gamma_{X_2}(1-\Gamma_C)}{1-\Gamma_C+\Gamma_{X_2}} = \frac{1-\Gamma_C}{1-\Gamma_C+\Gamma_{X_2}}.$$

It thus follows from (14) that the genuine gap size is

$$\frac{1}{1-p_0}\sum_{n=1}^{\infty} np_n = 1+\frac{1-\Gamma_{X_1}}{\Gamma_B}.$$

Note that this expression is independent of Γ_C and Γ_{X_2}.

4.3.4 Density of One-Bits

Let D be the total number of documents; we would like to relate this to N, the number of documents containing the designated term, given the model parameters. The predicted density of one-bits N/D, given the easily obtainable values N and D, allows a test of the model. We give two derivations of this important result.

Heuristic argument

To estimate N, note that we first have a span of zero or more documents without the designated term. This is followed by $N-1$ spans made up of a document containing the term followed by an inter-term span of zero or more documents. Finally, the last span consists of a document with the term, followed by a (possibly empty) terminal span. For our heuristic estimate, we assume that each internal span has a length equal to its expected value, and that the lengths of each of the terminal spans can be approximated as half of this length. So the D documents are made up of the N documents with a designated term, the two end spans and the $N-1$ internal spans. Therefore

$$D = N\sum_{n=0}^{\infty} np_n + N$$

$$= \quad N(1-\Gamma_C)\left(1+\frac{1-\Gamma_{X_1}}{\Gamma_B}\right)\left(\frac{1}{1-\Gamma_C+\Gamma_{X_2}}\right)+N.$$

This result immediately gives us an estimate for the density of 1-bits, N/D; this would serve as an estimate of the "p" value of an independence model, should we wish to simplify our model by using an independence approximation.

Formal derivation

One can also derive the above formula by observing that we are in states C or X_2 if and only if we have just generated a 1. Thus N, the number of ones, is equal to the number of recurrences of states C and X_2: $\pi_C D + \pi_{X_2} D = N$, which implies

$$\begin{aligned}
D &= \frac{N}{\pi_C+\pi_{X_2}} = N\frac{\Gamma_{X_2}\Gamma_B+2(1-\Gamma_C)\Gamma_B+(1-\Gamma_C)(1-\Gamma_{X_1})}{\Gamma_{X_2}\Gamma_B+(1-\Gamma_C)\Gamma_B}\\
&= N\frac{(1-\Gamma_C)\Gamma_B+(1-\Gamma_C)(1-\Gamma_{X_1})}{\Gamma_{X_2}\Gamma_B+(1-\Gamma_C)\Gamma_B}+N\\
&= N(1-\Gamma_C)\left(1+\frac{1-\Gamma_{X_1}}{\Gamma_B}\right)\left(\frac{1}{1-\Gamma_C+\Gamma_{X_2}}\right)+N,
\end{aligned}$$

agreeing with the preceding result.

4.3.5 Reductions to simpler models

By identifing X_2 with C, we get $\mathcal{M}$–3C. Now $\Gamma_{X_2}=\Gamma_C$ and we get

$$\sum_{n=0}^{\infty} np_n = (1-\Gamma_C)\left(1+\frac{1-\Gamma_{X_1}}{\Gamma_B}\right),$$

$$\sum_{n=1}^{\infty} n\frac{p_n}{1-p_0} = 1+\frac{1-\Gamma_{X_1}}{\Gamma_B}$$

and

$$D = N(1-\Gamma_C)\left(1+\frac{1-\Gamma_{X_1}}{\Gamma_B}\right)+N.$$

By identifying X_1 with B in $\mathcal{M}$–4S1, we get the complementary 3-state model $\mathcal{M}$–3B. Now $\Gamma_{X_1}=\Gamma_B$ and we get

$$\sum_{n=0}^{\infty} np_n = \frac{1-\Gamma_C}{\Gamma_B(1-\Gamma_C+\Gamma_{X_2})},$$

$$\sum_{n=1}^{\infty} n\frac{p_n}{1-p_0} = \frac{1}{\Gamma_B}$$

and

$$D = \frac{N(1-\Gamma_C)}{\Gamma_B(1-\Gamma_C+\Gamma_{X_2})}+N.$$

By identifing X_1 with B in $\mathcal{M}$–3C or by identifing X_2 with C in $\mathcal{M}$–3B we get the 2-state model. The formulas reduce to

$$\sum_{n=0}^{\infty} np_n = (1-\Gamma_C)\left(1+\frac{1-\Gamma_B}{\Gamma_B}\right) = \frac{1-\Gamma_C}{\Gamma_B},$$

$$\sum_{n=1}^{\infty} n\frac{p_n}{1-p_0} = 1+\frac{1-\Gamma_B}{\Gamma_B} = \frac{1}{\Gamma_B}$$

and

$$D = N\left(\frac{1-\Gamma_C}{\Gamma_B}\right) + N.$$

By identifying B with C in the 2-state model, we get the independence model. Now $\Gamma_B = \Gamma_C$ and let $\Gamma = \Gamma_B = \Gamma_C$. The formulae become

$$\sum_{n=0}^{\infty} np_n = \frac{1}{\Gamma} - 1,$$

$$\sum_{n=1}^{\infty} n\frac{p_n}{1-p_0} = \frac{1}{\Gamma}$$

and

$$D = \frac{N}{\Gamma}.$$

Model	Average genuine gap length $\frac{1}{1-p_0}\sum np_n$
indep	$1/\Gamma$
2-state	$1/\Gamma_B$
$\mathcal{M}$–3C	$1 + \frac{1-\Gamma_{X_1}}{\Gamma_B}$
$\mathcal{M}$–3B	$1/\Gamma_B$
$\mathcal{M}$–3S	$\frac{1-\Gamma_{X_1}+\Gamma_{X_1}\Gamma_B}{\Gamma_B(1-\Gamma_{X_1}+\Gamma_{X_1}^2)}$
$\mathcal{M}$–4S1	$1 + \frac{1-\Gamma_{X_1}}{\Gamma_B}$
$\mathcal{M}$–4S2	$\frac{1+\Gamma_B\Gamma_{X_2}-\Gamma_{X_1}}{\Gamma_B(1-\Gamma_{X_1}+\Gamma_{X_1}\Gamma_{X_2})}$
$\mathcal{M}$–4S3	$\frac{\Gamma_{X_1}\Gamma_B+(1-\Gamma_{X_1})(1-\Gamma_{X_2})}{\Gamma_B[\Gamma_{X_1}\Gamma_{X_2}+(1-\Gamma_{X_2})(1-\Gamma_{X_1}(1-\Gamma_{X_1}+\Gamma_{X_2}))]}$
$\mathcal{M}$–4C1	$1+(1-\Gamma_{X_1})\left(1+\frac{1-\Gamma_{X_2}}{\Gamma_B}\right)$
$\mathcal{M}$–4B1	$1/\Gamma_B$

Table 1: *Average genuine gap size for the different models*

4.4 Summary of the 4-state models

Using identical techniques to those of the previous section, one can derive similar formulae for the other 4-state models we considered earlier. The calculations are straightforward and therefore omitted. We report here on the resulting equations only for the symmetric 4-state models, for the C-biased model which performed best on most experiments ($\mathcal{M}$–4C1), and for its $\mathcal{C}$-symmetric complement ($\mathcal{M}$–4B1).

Model	D/N as function of model parameters
indep	$D = N/\Gamma$
2-state	$D = N\left(\frac{1-\Gamma_C+\Gamma_B}{\Gamma_B}\right)$
$\mathcal{M}$–3C	$D = N(1-\Gamma_C)\left(1+\frac{1-\Gamma_{X_1}}{\Gamma_B}\right)+N$
$\mathcal{M}$–3B	$D = \frac{N(1-\Gamma_C)}{\Gamma_B(1-\Gamma_C+\Gamma_{X_2})}+N$
$\mathcal{M}$–3S	$D = N(1-\Gamma_C)\left(\Gamma_{X_1}+\frac{1-\Gamma_{X_1}}{\Gamma_B}\right)\left(\frac{1}{1-\Gamma_C+\Gamma_C\Gamma_{X_1}}\right)+N$
$\mathcal{M}$–4S1	$D = N(1-\Gamma_C)\left(1+\frac{1-\Gamma_{X_1}}{\Gamma_B}\right)\left(\frac{1}{1-\Gamma_C+\Gamma_{X_2}}\right)+N$
$\mathcal{M}$–4S2	$D = N(1-\Gamma_C)\left(\Gamma_{X_2}+\frac{1-\Gamma_{X_1}}{\Gamma_B}\right)\left(\frac{1}{\Gamma_{X_2}+(1-\Gamma_C)(1-\Gamma_{X_1})}\right)+N$
$\mathcal{M}$–4S3	$D = N\frac{(1-\Gamma_C)[\Gamma_{X_1}\Gamma_B+(1-\Gamma_{X_1})(1-\Gamma_{X_2})]}{\Gamma_B[\Gamma_{X_1}\Gamma_{X_2}+(1-\Gamma_C)(1-\Gamma_{X_2})]}+N$
$\mathcal{M}$–4C1	$D = N(1-\Gamma_C)\left[1+(1-\Gamma_{X_1})\left(1+\frac{1-\Gamma_{X_2}}{\Gamma_B}\right)\right]+N$
$\mathcal{M}$–4B1	$D = \frac{N(1-\Gamma_C)}{\Gamma_B[\Gamma_{X_1}\Gamma_{X_2}+(1-\Gamma_C)(\Gamma_{X_2}+1)]}+N$

Table 2: *Density of 1-bits for different models*

The transition probabilities for models $\mathcal{M}$–4S2, $\mathcal{M}$–4S3, $\mathcal{M}$–4C1 and $\mathcal{M}$–4B1 can be summarized, respectively, by the following transition matrices, corresponding, as above, to the 4 states in the order (C, X_1, X_2, B):

$$T_{\mathcal{M}\text{–4S2}}=\begin{pmatrix}\Gamma_C & 1-\Gamma_C & 0 & 0\\ \Gamma_{X_1} & 0 & 0 & 1-\Gamma_{X_1}\\ \Gamma_{X_2} & 0 & 0 & 1-\Gamma_{X_2}\\ 0 & 0 & \Gamma_B & 1-\Gamma_B\end{pmatrix},\quad T_{\mathcal{M}\text{–4S3}}=\begin{pmatrix}\Gamma_C & 0 & 1-\Gamma_C & 0\\ 0 & 0 & \Gamma_{X_1} & 1-\Gamma_{X_1}\\ \Gamma_{X_2} & 1-\Gamma_{X_2} & 0 & 0\\ 0 & \Gamma_B & 0 & 1-\Gamma_B\end{pmatrix},$$

$$T_{\mathcal{M}\text{–4C1}}=\begin{pmatrix}\Gamma_C & 1-\Gamma_C & 0 & 0\\ \Gamma_{X_1} & 0 & 1-\Gamma_{X_1} & 0\\ \Gamma_{X_2} & 0 & 0 & 1-\Gamma_{X_2}\\ \Gamma_B & 0 & 0 & 1-\Gamma_B\end{pmatrix},\quad T_{\mathcal{M}\text{–4B1}}=\begin{pmatrix}\Gamma_C & 0 & 0 & 1-\Gamma_C\\ \Gamma_{X_1} & 0 & 0 & 1-\Gamma_{X_1}\\ 0 & \Gamma_{X_2} & 0 & 1-\Gamma_{X_2}\\ 0 & 0 & \Gamma_B & 1-\Gamma_B\end{pmatrix}.$$

We then procede by calculating the equilibrium probabilities satisfying $\pi = \pi T$, and evaluating the probability distributions of the run-lengths. Note that one ought to take care of the fact that the models differ slightly; for example, for $\mathcal{M}$–4S1 and $\mathcal{M}$–4S2, we are in state C or X_2 if and only if we have just generated a 1, while for $\mathcal{M}$–4S3 and $\mathcal{M}$–4B1, this is true for states C or X_1 or X_2, and for $\mathcal{M}$–4C1, only for state C. This has to be taken into account when evaluating p_n. Table 1 summarizes the formulae for the average genuine gap size $(\sum np_n)/(1-p_0)$ for the different models. For the sake of completeness, we have also included the independent, 2-state and 3-state models.

Making the same assumptions as above about the spans of zeroes, we can derive the following relationships between D and N, which are summarized in Table 2.

5. Experimental Evaluation of Models

5.1 Model-based coding algorithm

The following algorithm is a formal description of the encoding process for any of the Markov models discussed above. Given is a bit-vector of length n bits: $b_1 \cdots b_n$. First the model $\mathcal{M} = \langle Q, \delta_{\mathcal{M}}, S \rangle$ of the transitions is chosen, where Q is the set of states, $S \in Q$ is the starting state, and $\delta_{\mathcal{M}} : Q \times \{0,1\} \longrightarrow Q$ is the corresponding transition function. A two-dimensional array count$[i,j]$ with $i \in Q$ and $j \in \{0,1\}$ is used to keep track of the number of times each of the possible transitions has occurred. The function $encode(b,p)$ is part of an arithmetic encoder applied to a binary alphabet $\{0,1\}$, where the probability of a 1 is p, and the bit b is to be encoded, i.e., the current sub-interval $I \subseteq [0,1]$ is partitioned according to p, and the new sub-interval $I' \subseteq I$ is the one corresponding to b.

<u>Model–$\mathcal{M}$ coding</u>

```
{
        Choose model M

                /* start in state S and collect statistics */
        state ⟵ S
        for i ⟵ 1 to n
        {
                count[state, b_i] ⟵ count[state, b_i] + 1
                state ⟵ δ_M(state, b_i)
        }
                /* calculate transition probabilities */
        for all j ∈ Q
                δ_j ⟵ count[j,1] / (count[j,1]+count[j,0])

                /* second scan for actual encoding */
        state ⟵ S
                /* initialize lower and upper limits of output-interval */
        I_ℓ ⟵ 0;   I_u ⟵ 1
        for i ⟵ 1 to n
        {
                encode (b_i, δ_state)
                state ⟵ δ_M(state, b_i)
        }
        output any x ∈ [I_ℓ, I_u]
}
```

encode (b, p)
{

if $b = 1$ **then**

$$I_u \longleftarrow I_\ell + p\,(I_u - I_\ell)$$

else

$$I_\ell \longleftarrow I_\ell + p\,(I_u - I_\ell)$$

}

The parameter evaluation algorithm for the HMM relies on three control-parameters, (i, r, t). Recall that the parameters of the HMM cannot be solved directly, and that an iterative maximum likelihood estimation procedure is required. Such procedures generally lead to local, not global, optima. We thus procede as follows: For each bit vector we form i initial models; for each model we undertake r iterations of our estimation procedure. We select the model giving the best compression, and then continue the iterations with the chosen model until the compression gain between two successive repetitions drops below a predetermined threshold value t. At this point we have the final model for the current bit vector, and the parameters (which are the four probabilities $A(0,0), A(1,0), B(0,0), B(1,0)$) and the encoding result are stored; then follows the encoding phase using that model.

5.2 Parameter Estimation

We can estimate the parameters directly. We assume that before generating the bitmap, we are in state B. The sequence of one's and zero's making up the bitmap completely determines the state at any bitmap position. Thus it is easy to tabulate the number, and hence probability, of each type of transition.

For illustration, consider the following bitmap, as processed by the Markov model, $\mathcal{M}$–3C:

$$0\;0\;1\;0\;1\;1\;0\;0\;\cdots$$

Since we begin in state B, the initial zero indicates that the first transition is back to state B. Continuing in this manner we find the sequence of states corresponding to the above bitmap is given as follows (with the initial B preceding the colon):

$$B:\;B\;B\;C\;X\;C\;C\;X\;B\;\cdots$$

In this sequence, we are in state B four times, for which we have three transitions. Of these three transitions, one is to state C, so on the basis of the information given, we would estimate $\Gamma_B \approx 1/3$; similarly $\Gamma_C \approx 1/3$, and $\Gamma_X \approx 1/2$. Thus each parameter is easily evaluated and these parameters can be used as the basis for compressing the bitmap. Further, the standard deviation of this estimate, for state S, is given by $\sigma_S = \sqrt{\Gamma_S(1-\Gamma_S)/N_S}$, if we experience N_S transitions from state S in the pertinent bitmap. The standard deviations make it possible to compute confidence intervals and do tests of hypotheses.

For a large bitmap, the Γ-values can be stored with the bitmap to permit decompression. But several mechanisms can be tried to reduce the cost of storing these parameters. For example, we can save the space for storing these parameters by evaluating them adaptively, beginning with reasonable initial values.

We can also lower storage costs by reparameterizing the model: The Γ's are our model's basic parameters. But we can also consider a reparameterization that is suggestive: $\Gamma_C \equiv \Gamma$, $\Gamma_X \equiv \theta_X \Gamma_C$, and $\Gamma_B \equiv \theta_B \Gamma_X$, which define the parameters Γ, θ_X, and θ_B. These parameters satisfy simpler constraints ($0 \le \theta_X, \theta_B \le 1$) independently of one another and of Γ, which will ease problems of estimation. We also expect that regularities in the data will be more easily expressed in terms of the θ's than in terms of the Γ's. It is also useful to define $\Theta \equiv \theta_X \theta_B$, so that $\Gamma_B = \Theta \Gamma_C$. Θ is a single value reflecting the strength of clustering: it indicates the relative likelihood of a term appearing if we are inside a cluster as compared to if we are outside a cluster.

This reparameterization allows us to lower storage costs if we find the θ's are related in a regular manner over the terms. For example, since state X is intermediate between states C and B, we might find that Γ_X is reasonably approximated by the average of Γ_C and Γ_B, that is, that $\theta_X = (1+\Theta)/2$.

It may also be possible, without serious deterioration in performance, to divide the parameters into a small number of categories. Thus, we might divide our terms into four clustering classes: one class would represent no clustering ($\Theta = 1$); the other three states would represent varying clustering strengths, with the values of the clustering parameter, Θ, for these states determined empirically. This simplification allows clustering strength to be represented with the cost of just two bits per term.

5.3 Performance Estimates

The availability of a detailed model permits us to estimate model performance. For illustration, we will establish some properties for model $\mathcal{M}$–4S1. If we are in state $S \in \{B, X_1, X_2, C\}$, we expect that we can encode the next bitmap element in $H(\Gamma_S)$ bits (where $H(x) \equiv -(x \log(x) + (1-x)\log(1-x))$). Using eqs. for π_C, π_{X_1}, π_{X_2}, and π_B, we estimate that the D bits of the bitmap can be reduced to

$$B_M = (\pi_C H(\Gamma_C) + \pi_B H(\Gamma_B) + \pi_{X_1} H(\Gamma_{X_1}) + \pi_{X_2} H(\Gamma_{X_2}))D$$

bits, if the model is valid and we base our codes on that model.

This estimate of performance can be used as a rough test of the Markov model. But even if the model is valid, we are left with the question of whether the savings of using the full model justifies the additional complexity relative to, say, the independence model. The relative performance of the two models is easily computed.

Under the assumption of the independence model, the appearance of a one-bit is determined by a single parameter, p, the density of one-bits. Developing a code using the value p and assuming a zeroth order model, the size of the bitmap is reduced to

$$B_I = H(p)D$$

bits. But if the true model is $\mathcal{M}$–4S1, we can compute the value we would find for p from the model parameters, and thereby predict the performance we would get if we pretended the independence model was valid. The density of one-bits is just the probability of a random bit being set to one. The model predicts that the probability of a one-bit is given by:

$$\begin{aligned} p &= \pi_B \Gamma_B + \pi_{X_1} \Gamma_{X_1} + \pi_{X_2} \Gamma_{X_2} + \pi_C \Gamma_C \\ &= \pi_C + \pi_{X_2}, \end{aligned}$$

the last equality following from the equilibrium equations. The equation $p = \pi_C + \pi_{X_2}$ could have been asserted directly: the probability of a one-bit is just the probability of going to state C from any of the states, or going to the state X_2; but the long term probability of *going* to these states is the same as the probability of *being* in them, as the last equation asserts.

The ratio B_M/B_I gives the relative advantage of using the full Markov model vs using the independence model even if the true model is more complex.

5.4 Experimental results

In our earlier papers we used the Bible as "database", its chapters acting as documents. Because of its small size, this choice was useful for testing purposes, as well as providing performance data for an interesting database. We can now provide performance statistics for two of the world's largest natural language IRS's: the TLF, a database of 680 MB of French language texts (112 million words) of the 17^{th}–20^{th} centuries [6], and the database of the RRP, 350 MB of Hebrew and Aramaic texts (60 million words) written over the past ten centuries [14]. Their uncompressed concordances span about 345 MB for TFL (excluding references to the 100 most frequent words, considered as *stop-words*), and 450 MB for the Responsa database, for which each word is referenced. In fact, we took only sub-collections of these: for TLF, the 35070 terms belonging to the (lexicographic) range between `elle` and `flaube`, without limiting the document range, so that each uncompressed bitmap had length $D = 38757$ bits; for RRP we took all the 300000 different terms, but restricted the document range to include only the $D = 8119$ documents written in the 20^{th} century. In order to allow comparisons with the methods of the earlier papers, we include also results on the Bible.

Table 3 gives general statistics on the tested files and summarizes the results. The column corresponding to the English Bible (King James Version) uses the same restriction as in our earlier work, i.e., only words appearing in at least 60 documents are tested. For real-life databases, most of the words occur rarely, so their bitmaps will be encoded by simply enumerating the 1-bits (see [8]). We thus decided, for TLF and RRP, to consider only words that appear in at least 0.2% of the possible documents, and to partition the terms into three classes, according to the number of documents N in which the terms occur. The classes are: $N \in [0.2\% - 1\%)$, $N \in [1\% - 3\%)$ and $N \in [3\% - 100\%]$. The threshold values thus were 78, 388 and 1162 for TLF and 16, 81 and 244 for RRP.

The upper part of Table 3 shows, for each class, the number of different terms, and their total number of occurrences. In the lower part of the table, each line corresponds to one of the methods discussed above. To understand the values in the table, recall that we are representing the top level of the concordance as follows: for each term, we list sequentially the documents in which the term occurs. As our measure of compression for the list corresponding to a term, we compute the number of bits needed to encode this list with our methods, and divide this value by the number of documents in which the term occurs. The table gives the average of this quantity for all the terms in a class. In other words, it is the average, per 1-bit, of the number of bits needed to encode the 1-bits of all the bitmaps in this class. The results for HMM correspond to the estimation parameters $(10, 1, 0.001)$, that is, the best out of 10 initial models, each after a single iteration, is chosen, and then improved until the relative gain between successive runs falls below 0.001. The 4- and 3-state models are referred to by their names. The independence model is the 1-state Markov model used in [3]. The row entitled *run-length* gives the best result out of a range of different run-length encoding schemes, including Elias' γ and δ codes [10], various Exp-Golomb codes [23] and Start-Step-Stop codes [13].

As can be seen for the files for which HMM results are listed, they usually were best among all the tested methods. However, just to produce the numbers on the TLF, more than 6 days of CPU were necessary (as compared to at most one hour for all the other tests). This also explains why not all the values are given. The best values for the other methods are framed. We see that on all the real concordance files, the C-biased model $\mathcal{M}$–4C1 performed best, except for the lowest density bitmaps, for which $\mathcal{M}$–4C4 was slightly better. Moreover, when ordering the methods by compression efficiency, similar rankings were obtained, with all the C-biased methods performing better than any of the B-biased ones.

We have not included the cost of storing the necessary parameters, which is negligible in most of the cases. Four probabilities need be stored for the 4-state models and 3 for the 3-state models. If we choose the best out of a possible set of 8 probabilities (as in [4]), then only 3 bits are needed to store the parameters for each term. But since only the high frequency terms are to be compressed, the average number of added bits per term occurrence is very small. For the ranges given in Table 3

Database:	Bible	TLF			RRP		
	60–929	78–387	388–1162	1162–∞	16–80	81–243	244–∞
terms	623	2032	619	381	28039	7421	4631
occurrences	131874	352522	402890	1387698	967543	1008365	3962924
HMM	2.490		6.62	3.39			
run-length	2.923	8.60	6.76	3.71	9.18	7.13	4.06
independent	2.683	9.09	7.26	4.02	9.17	7.27	3.95
2-state	2.593	8.72	6.88	3.66	8.92	7.01	3.78
$\mathcal{M}$–3C	2.570	8.57	6.73	3.56	8.86	6.94	3.72
$\mathcal{M}$–3B	2.579	8.70	6.86	3.62	8.91	6.98	3.76
$\mathcal{M}$–3S	2.560	8.65	6.80	3.55	8.90	6.96	3.71
$\mathcal{M}$–4S1	2.555	8.55	6.71	3.52	8.85	6.92	3.70
$\mathcal{M}$–4S2	2.555	8.65	6.80	3.54	8.89	6.96	3.71
$\mathcal{M}$–4S3	2.544	8.64	6.78	3.51	8.90	6.95	3.69
$\mathcal{M}$–4C1	2.557	$\boxed{8.48}$	$\boxed{6.65}$	3.51	$\boxed{8.81}$	$\boxed{6.89}$	3.69
$\mathcal{M}$–4C2	2.555	8.55	6.71	3.52	8.84	6.92	3.70
$\mathcal{M}$–4C3	2.544	8.52	6.68	3.48	8.84	6.90	3.67
$\mathcal{M}$–4C4	$\boxed{2.544}$	8.51	6.67	$\boxed{3.48}$	8.84	6.90	$\boxed{3.67}$
$\mathcal{M}$–4C5	2.546	8.62	6.76	3.51	8.89	6.94	3.68
$\mathcal{M}$–4B1	2.572	8.70	6.86	3.61	8.91	6.98	3.75
$\mathcal{M}$–4B2	2.561	8.66	6.81	3.56	8.89	6.96	3.72
$\mathcal{M}$–4B3	2.552	8.65	6.79	3.54	8.90	6.96	3.71
$\mathcal{M}$–4B4	2.560	8.68	6.83	3.57	8.90	6.97	3.73
$\mathcal{M}$–4B5	2.556	8.68	6.83	3.56	8.91	6.97	3.71

Table 3: *Statistics and compression results*

for TLF and RRP, it is always below 1%, except for the low frequency Responsa terms, for which it reaches 4% for the 4-state models.

When we began these experiments, we were concerned that our results might be very database sensitive, since both language and, at the highest level of the concordance with which we are now dealing, the database organization would be expected to influence the extent of clustering. We therefore were surprised to note the extent of similarity between the TLF and RRP results: within the same range of term density, the results for a given method differ by only 1–5%, with compression on TLF being consistently better. We therefore conclude that the main factor acting on the compression efficiency is the 1-bit density of the bitmaps, rather than language dependent features. As expected, compression improves with increasing density.

We are conceptualizing the concordance as indexing term occurrences at a variety of levels, for example, document or section or sentence. Most of our results reflect compression at the document level. The Bible results give us insight into how our algorithms perform at lower levels of the hierarchy, since here we are considering occurrences of terms over text segments of a single document. This observation, together with the fact that the Bible maps were much denser (at least 6.4% 1-bits), may explain why the results for the Bible differ so markedly from that of the Responsa and TLF databases.

Note also that the best results improve upon those obtained from the independence model by 5–13%, which is not negligible, considering the sizes of the databases. There is also a clear tendency, both for TLF and RRP, of having greater improvement with increasing density. We thus conclude that the new models presented here may indeed represent a genuine improvement in our ability to compress concordances.

6. Conclusion

The main objective of this paper has been to develop models that describe the dependencies among term occurrences in text in order to create codes that compress concordances more effectively than those based on the usual assumption of term independence. The assumption that governed our considerations was the existence of an underlying Hidden Markov Model, or HMM. We did indeed find a small, but significant, improvement in compression effectiveness when using an HMM. However, such models make great demands on computational resources.

To limit these costs, we approximated the HMM by traditional Markov models. A set of criteria we developed allowed us to greatly reduce the very large set of candidate n-state models, making it possible to provide a full inventory for $n \leq 4$. Further simplification is provided by graph theoretic *reduction* and *complementation* operations defined among the various models. These were used to provide a structure relating the models studied, and may well find application in other graph based problems. Finally, tests on the concordances of the English Bible and of two of the world's largest full-text retrieval system: the Trésor de la Langue Française and the Responsa Project, demonstrated that traditional Markov models allowed great improvements in computational efficiency, at the cost of very little deterioration of compression performance.

The concordance remains a large and still relatively unstudied component of Information Retrieval Systems. This paper concentrated on models of localized clustering. But with sparse bitmaps, such models prematurely leave a cluster or between-cluster state. We could compensate for this by introducing more transitional states; instead further work is being planned in which models which better retain the memory of which state it is in are developed.

Appendix

Equivalence of encoding bits and run lengths

We have contemplated two different ways of encoding a bitmap: we can encode each bit in turn, using arithmetic encoding to attain the entropy limit; or we can count the number of zeroes between ones, and encode the gap size. One would expect these to be equivalent. Here we argue this is the case.

We assume a state model. Suppose we have just encoded a one-bit (or are at the beginning of the bitmap) and are in state S_0. Let us compute the number of bits required to encode the next sequence of bits up to and including the next one-bit, or until we reach the end of the bitmap.

To establish our notation: If we are in state S_i and generate a zero-bit, we make a transition into state S_{i+1}. If we are in state S_i, the probability of generating a one-bit is Γ_i. Note that we are not making any assumptions on the memory-size of the model.

Suppose we have D bits left to encode. Then the probability of a string of r zeroes is given as follows:

r	P_r
0	Γ_0
1	$(1-\Gamma_0)\Gamma_1$
2	$(1-\Gamma_0)(1-\Gamma_1)\Gamma_2$
	$\cdots$
$D-1$	$(1-\Gamma_0)(1-\Gamma_1)\cdots(1-\Gamma_{D-2})\Gamma_{D-1}$
D	$(1-\Gamma_0)(1-\Gamma_1)\cdots(1-\Gamma_{D-2})(1-\Gamma_{D-1})$

Thus the expected length to encode the r value is:

$$\begin{aligned}
H \;=\; & -\Gamma_0\log(\Gamma_0) \\
& -(1-\Gamma_0)\Gamma_1\log\left((1-\Gamma_0)\Gamma_1\right) \\
& -\cdots \\
& -(1-\Gamma_0)(1-\Gamma_1)\cdots(1-\Gamma_{D-2})\Gamma_{D-1}\log\left((1-\Gamma_0)(1-\Gamma_1)\cdots(1-\Gamma_{D-2})\Gamma_{D-1}\right) \\
& -(1-\Gamma_0)(1-\Gamma_1)\cdots(1-\Gamma_{D-2})(1-\Gamma_{D-1})\log\left((1-\Gamma_0)(1-\Gamma_1)\cdots(1-\Gamma_{D-2})(1-\Gamma_{D-1})\right)
\end{aligned}$$

Expanding the logarithms, we find:

$$\begin{aligned}
H \;=\; & \Gamma_0[-\log\Gamma_0] \\
& +(1-\Gamma_0)\Gamma_1[-\log(1-\Gamma_0)-\log(\Gamma_1)] \\
& +\cdots \\
& +(1-\Gamma_0)(1-\Gamma_1)\cdots(1-\Gamma_{D-2})\Gamma_{D-1}[-\log(1-\Gamma_0)-\log(1-\Gamma_1)-\cdots \\
& \qquad -\log(1-\Gamma_{D-2})-\log(\Gamma_{D-1})] \\
& +(1-\Gamma_0)(1-\Gamma_1)\cdots(1-\Gamma_{D-2})(1-\Gamma_{D-1})[-\log(1-\Gamma_0)-\log(1-\Gamma_1)-\cdots \\
& \qquad -\log(1-\Gamma_{D-2})-\log(1-\Gamma_{D-1})]
\end{aligned}$$

But note that if we are in state S_i, it requires $-\log(1-\Gamma_i)$ bits to encode a single zero-bit and $-\log(\Gamma_i)$ bits to encode a single one-bit. Thus we see that the terms in brackets are just the total number of bits that would be required to encode the string of zero-bits followed by a one-bit (or terminating zero-bit) for the string represented by the term, if we encode a single bit at a time. Thus H is also the expected value of the code-size of the next run of zero-bits, encoding each bit individually, as was to be shown.

The second form can be reorganized to give a very nice alternative expression for H. To do this, we bring together terms with the same $\log\Gamma_i$ and $\log(1-\Gamma_i)$:

$$H = -\Gamma_0\log\Gamma_0 - (1-\Gamma_0)\log(1-\Gamma_0)\,[\Gamma_1 + (1-\Gamma_1)\Gamma_2 + (1-\Gamma_1)(1-\Gamma_2)\Gamma_3 + \cdots$$

$$
\begin{aligned}
&\quad +(1-\Gamma_1)(1-\Gamma_2)\cdots(1-\Gamma_{D-1})\Gamma_D + (1-\Gamma_1)(1-\Gamma_2)\cdots(1-\Gamma_{D-1})(1-\Gamma_D)] \\
&- \cdots - \\
&(1-\Gamma_0)(1-\Gamma_1)\cdots\Gamma_{D-2}\log\Gamma_{D-2} - (1-\Gamma_0)(1-\Gamma_1) \\
&\quad \cdots(1-\Gamma_{D-2})\log(1-\Gamma_{D-2})\,[\Gamma_{D-1} + (1-\Gamma_{D-1})] \\
&-(1-\Gamma_0)(1-\Gamma_1)\cdots\Gamma_{D-1}\log\Gamma_{D-1} - (1-\Gamma_0)(1-\Gamma_1) \\
&\quad \cdots(1-\Gamma_{D-1})\log(1-\Gamma_{D-1}).
\end{aligned}
$$

Each sum of terms in brackets equals one (as can be seen by telescoping backwards), leaving an expansion of the form:

$$H_0 + (1-\Gamma_0)H_1 + \cdots + (1-\Gamma_0)(1-\Gamma_1)\cdots(1-\Gamma_{D-2})H_{D-1},$$

where H_i is just the entropy $-\Gamma_i\log\Gamma_i - (1-\Gamma_i)\log(1-\Gamma_i)$, which is the expected number of bits required to encode the next bit position in state S_i, provided we get that far. But each H_i is multipled by the probability of indeed getting that far, so the sum cummulates the expected contributions to the encoding of the next run of zeroes of each bitmap position.

References

[1] **Asai, K., Hayamizu, S., Onizuka, K.** HMM with Protein Structure Grammar, *Proc. 26th Hawaii Int. Conf. on System Sciences* (1993) 783-791.

[2] **Bell T.C., Witten I.H., Cleary J.G.,** Modeling for text compression, *ACM Computing Surveys* **21** (1989) 557–591.

[3] **Bookstein A., Klein S.T., Raita T.,** Model based concordance compresssion, *Proc. Data Compression Conference DCC–92*, Snowbird, Utah (1992) 82–91.

[4] **Bookstein A., Klein S.T., Raita T.,** Markov models for clusters in concordance compression, *Proc. Data Compression Conference DCC–94*, Snowbird, Utah (1994) 116–125.

[5] **Kemeny, J.G. and Snell, J.L.,** *Finite Markov Chains*, New York: Van Nostrand (1960).

[6] **Bookstein A., Klein S.T., Ziff D.A.,** A systematic approach to compressing a full text retrieval system, *Information Processing & Management* **28** (1992) 795–806.

[7] **Choueka Y., Fraenkel A.S., Klein S.T.,** Compression of Concordances in Full-Text Retrieval Systems, *Proc. 11-th ACM-SIGIR Conf.,* Grenoble (1988) 597–612.

[8] **Choueka Y., Fraenkel A.S., Klein S.T., Segal E.**, Improved Hierarchical Bit-Vector Compression in Document Retrieval Systems, *Proc. 9-th ACM-SIGIR Conf.,* Pisa (1986) 88–97.

[9] **Cormack G.V., Horspool R.N.,** Data compression using dynamic Markov modelling, *The Computer Journal* **30** (1987) 541–550.

[10] **Elias P.,** Universal codeword sets and representation of the integers, *IEEE Trans. on Inf. Th.,* **IT–12** (1975) 194–203.

[11] **Even S.,** *Algorithmic Combinatorics*, The Macmillan Company, NY (1973).

[12] **Feller W.,** *An Introduction to Probability Theory and its Applications*, Wiley, New York (1968).

[13] **Fiala E,R., Greene D.H.,** Data compression with finite windows, *Comm. of the ACM* **32** (1989) 490–505.

[14] **Fraenkel A.S.,** All about the Responsa Retrieval Project you always wanted to know but were afraid to ask, expanded summary, *Jurimetrics J.* **16** (1976) 149–156.

[15] **Haussler D., Krogh A., Mian S., Sjölander K.,** Protein Modeling using Hidden Markov Models: Analysis of Globins, *Proc. 26th Hawaii Int. Conf. on System Sciences* (1993) 792-802.

[16] **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.

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

[18] **Linoff G., Stanfill C.,** Compression of indexes with full positional information in very large text databases, *Proc. 16-th ACM-SIGIR Conf.,* Pittsburgh (1993) 88–95.

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

[20] **Mittendorf E., Schäuble P.,** Document and passage retrieval based on Hidden Markov models, *Proc. 17-th ACM-SIGIR Conf.,* Dublin (1994) 318–327.

[21] **Moffat A., Zobel J.,** Coding for compression in full-text retrieval systems, *Proc. Data Compression Conference DCC–92*, Snowbird, Utah (1992) 72–81.

[22] **Rabiner L.R.,** A tutorial on hidden Markov models and selected applications in speech recognition, *Proc. IEEE*, **77**, **2**, (Feb. 1989) 257–286.

[23] **Teuhola J.**, A compression method for clustered bit-vectors, *Inf. Proc. Letters* **7** (1978) 308–311.

[24] **Wisniewski J.L.,** Compression of index term dictionary in an inverted file oriented database: some efficient algorithms, *Information Proc. & Management* **22** (1986) 493–501.

[25] **Witten I.H., Bell T.C., Nevill C.G.,** Models for compression in full-text retrieval systems, *Proc. Data Compression Conference DCC–91*, Snowbird, Utah (1991) 23–32.

[26] **Witten I.H., Moffat A., Bell T.C.,** *Managing Gigabytes, Compressing and Indexing Documents and Images*, International Thomson Publishing, London (1994).