in *Information Processing & Management* **28** (1992) 735–748

Models of Bitmap Generation: A Systematic Approach to Bitmap Compression*

Abraham Bookstein[1], *Shmuel T. Klein*[2]

[1]Center for Information and Language Studies, University of Chicago, Chicago IL 60637, USA

[2]Dept. of Math. & CS and Dept. of Economics & BA, Bar Ilan University, Ramat Gan 52900, Israel

Abstract: In large IR systems, information about word occurrence may be stored in form of a bit matrix, with rows corresponding to different words and columns to documents. Such a matrix is generally very large and very sparse. New methods for compressing such matrices are presented, which exploit possible correlations between rows and between columns. The methods are based on partitioning the matrix into small blocks and predicting the 1-bit distribution within a block by means of various bit generation models. Each block is then encoded using Huffman or arithmetic coding. The methods also use a new way of enumerating subsets of fixed size from a given superset. Preliminary experimental results indicate improvements over previous methods.

1. Introduction

The common approach to processing complex boolean queries in large full-text document retrieval systems is to use inverted files: a concordance is accessed via a dictionary, and includes for each different word of the text, the ordered list of exact references to its occurrences. To process the query A_1 AND A_2 AND ... AND A_m, where A_i stands for a disjunction of keywords $A_i \equiv (A_{i1}$ OR A_{i2} OR ... OR $A_{in_i})$, we perform a sequence of unions and intersections, $\bigcap_{i=i}^{m}\left(\bigcup_{j=1}^{n_i} L(A_{ij})\right)$, where $L(X)$ is the list of "coordinates" locating occurrences of the keyword X.

Alternatively, one could use a set of *bitmaps* which act as "occurrence maps". For each different word X of the text, a vector $B(X)$ is constructed; all the vectors $B(X)$ have the same length, which is the number of documents (or retrieval units) in the system; the bit in position i of $B(X)$ is turned on (set to 1) if and only if X appears in document number i. One of the advantages of this approach is that processing complex queries now reduces to performing logical AND and OR operations on bitstrings, which is easily done on most machines. For the query above, one evaluates a new bitmap $F = \bigwedge_{i=i}^{m}\left(\bigvee_{j=1}^{n_i} B(A_{ij})\right)$; the set of indices of the 1-bits of F is then the requested set of relevant documents.

The main problem of the bitmap approach is the huge amount of storage required by the bitmaps. Fortunately, these bitmaps tend to be very sparse, that is, the number of 1-bits is generally substantially smaller than the number of 0-bits. Thus various compression methods may be applied. Depending on the density of 1-bits in the maps, compression

*This paper combines and extends preliminary versions that were presented at the DCC–91 Conference in Snowbird, Utah, and at the SIGIR–91 Conference in Chicago.

rates of up to 95% have been reported. Techniques for compressing bitmaps include variants of run-length coding [17], [18], Huffman coding [12], [10], and hierarchical methods [19], [6]. All of these try to compress each bitmap independently from the others. In [3], a new method is introduced that attempts to improve compression efficiency of a set of bitmaps by collecting similar maps into *clusters*.

In the current work, we try a different approach to the problem of compressing a *set* of bitmaps (in fact a bit table). Often, in such bitmap sets, bits in nearby columns are related to one another, and similarly for corresponding bits in successive rows. While each row of the table acts as occurrence map for the given word, a column can be viewed as representing a dictionary for the given document. If so, bits in rows corresponding to frequently occurring words are more likely to have been turned on than those in rows corresponding to rarer words. Similarly, in columns associated with longer documents, the probability of a bit being on is greater than for a shorter document. In this paper, which extends the work in [4] and [5], we are exploring the possibility of exploiting the structure between as well as within bitmaps to compress the whole bit-table, using simple models of bit occurrence. The strategy of separating model construction and compression method continues the now well established practice of basing compression on an explicit model of message generation [2].

2. Description of the compression technique

The basic idea of all the methods described below is to partition the rows of the table into blocks of fixed size N bits and to estimate the probabilities of the different bit-blocks by means of the assumed model. Once we have formulated the problem as encoding a set of objects for which the probability of occurrence is known, the use of Shannon-Fano, Huffman (see [11]) or arithmetic coding [20] is suggested.

Basically, our method encodes fixed sized blocks. However, as an elaboration, we recognize that the block containing only zeros might be the most common, and thus often we will have a sequence of zero blocks. If this is the case, then we could encode 00, 000, etc., where multiple zeros denote sequences of null blocks. Note that if this procedure is used, we must modify the probability distribution of the block following the zero-block sequence, since the zero-block itself cannot appear. In effect we are integrating run length coding into the framework of Shannon-Fano or Huffman codes, offering a more efficient encoding of the run lengths and improving the encoding of non-zero bit segments as well. This can be done for arithmetic encoding as well, but is likely to be more useful for the Shannon-Fano and Huffman approaches, which suffer when probabilities near one appear.

Jakobsson [12] has already suggested the use of Huffman coding for bitmap compression, using empirical counts of patterns rather than a model to define the probabilities. However, his method is limited in terms of the size N of the blocks that can be compressed, because codewords for all of the 2^N possible bit-patterns are generated. But given a model of bitmap generation, the following simple argument suggests how larger blocks can be Huffman encoded. Consider the process by which objects are encoded using Huffman's algorithm. We first arrange the objects in a list by decreasing values of probabilities. If we do so, and assume bits are turned on locally independently of one another, then all blocks with the same number of bits turned on will appear together in this list. Then, as we combine objects into clusters for encoding, those objects with the same number of bits on will tend to remain together. Continuing in this manner, at one point we will have a Huffman tree, where each node represents a set of blocks, and where all the blocks within a set have the same number of bits turned on. The full Huffman tree will continue by representing each actual block under the node encoding the set it is contained in. Since all the blocks within a set have the same number of bits turned on, the are nearly equi-probable, so the within-set representation is effectively an enumeration of the blocks within the set.

Thus to encode a segment, we could first create codewords for the numbers k equal 0 to N (possibly including 00, 000, etc) based on the probabilities of occurrence of a block with k bits on. Then we encode the block by encoding k, the number of bits on, followed by the codeword for the particular pattern of k bits. But there will typically be a very large number of blocks for a given value of k. If we are able to provide an enumeration of all such bit patterns, each pattern can be encoded by the binary representation of its location in the enumeration. Many such enumerations exist. In the next section, we derive a new enumeration method, specially adapted to our problem by permitting fast decoding with a small storage overhead. A reader interested only in the modeling component of our method can skip to Section 4.

3. A new enumeration method for subsets

Suppose that our bitmaps are stored as blocks of N bits, of which k bits are turned on, for $k \ll N$. N bits are needed for the most direct way to store a block. Alternatively, for small k we might be better off by indicating the position of each 1-bit within the block individually, using $k\lceil \log_2 N \rceil$ bits. But both of these are inefficient ways of storing sparse bitmaps: there are only $\binom{N}{k}$ possible blocks for which k bits are on, so representing the given bitmap by its index in an enumeration of all bitmaps with k bits on requires only $\lceil \log \binom{N}{k} \rceil$ bits of storage. If N equals 32 and typically only two bits are turned on, we would need $2 \log_2 32 = 10$ bits to list the two positions of the 1-bits individually, but only $\lceil \log_2 \binom{32}{2} \rceil = 9$ bits by means of an enumeration.

There are many ways to enumerate a set of bitmaps [7]. For example, since a block of N bits with k 1-bits can be interpreted as representing a subset of k elements from a given superset of N elements, we can adopt any subset enumeration method. The one reported by Reingold & al. [15] is both elegant and captures the natural lexicographic ordering of the subsets. But in some applications, an enumeration that expresses a different ordering may be beneficial. If so, a representation should be developed that reflects the needs of the problem at hand. For our problem of bitmap storage, we note that often the 1-bits tend to cluster — that is, if a bit is on, the likelihood of the following bit being turned on is increased. If this is the case, certain k-subsets are more likely to occur than others, and this variability in probability can be exploited to produce a more efficient representation. For example, if the order induced by our enumeration corresponds to decreasing probability of occurrence of the bit-blocks, we could use a *universal* representation of the integers to encode the index in the enumeration (see for example [8] or [1]). A universal representation assigns shorter codewords to small integers and longer codewords to large integers, achieving thereby a more compact encoding than by using a fixed length representation. But a tendency to cluster is common in sets occurring in many problem areas, so an enumeration that recognizes this can have general applicability. For such problems, we would thus like to enumerate the k-subsets in a manner that brings together subsets having a comparable degree of clustering.

To procede, we need a measure of bitmap clustering. Suppose then that as we go from left to right on the bitmap, we encounter a string (possibly null) of zero-bits, followed by a sequence of bits beginning with the leftmost 1-bit of the bitmap and ending with the rightmost 1-bit of the bitmap (we shall refer to this sequence as the *pattern*, p, of the bitmap), then terminating with a possibly null seuqence of zero-bits. Further, a bitmap can be defined by describing its pattern, then indicating where in the bitmap the pattern begins. Thus the bitmaps 10100 and 01010 are different, but have the same pattern.

We shall measure the degree of clustering of a bitmap by the diameter, $D(p)$, of its pattern p: the number of bits between and including its leftmost and rightmost 1-bits. This is the sum of k, the number of bits turned on, and the number of zeros between the extreme 1-bits. Our task, then, is to enumerate the bitmaps in such a manner that if $D(p_1) < D(p_2)$ then the bitmap described by p_1 appears before that described by p_2 in the enumeration, that is, the bitmap described by p_1 has the smaller index.

3.1 Theory

The first couple of cases are easy to enumerate. For k equal to zero, it is useful to assign the bitmap the value zero. If k equals one, then the value assigned to the bitmap is the value, between one and N, of the location of the only 1-bit.

We thus need consider only patterns of k 1-bits within a bitmap of N bits, for $k \geq 2$. We conceptualize such a bitmap as being a pattern of diameter D shifted to its place within the bitmap. The pattern consists of k 1's and $D - k$ zeros. To be consistent with our constraint, all bitmaps having patterns with smaller D must have appeared earlier. For $D \geq 2$, there are $\binom{D-2}{k-2}$ different patterns of diameter D, since the D locations constituting a pattern must begin and end with a 1-bit; thus, to define the pattern we need only choose the locations of the remaining $k-2$ 1-bits from the $D-2$ locations still eligible.

Since a pattern of diameter d can occur in any of $N-d+1$ positions within the background bitmap, we conclude that bitmaps with patterns having diameter D begin with index $\sum_{d<D}\binom{d-2}{k-2}(N-d+1)$ within the enumeration.

To continue the definition of our enumeration, we now consider the relative order of bitmaps having equal diameter (that is, whose patterns have the same diameter D). If two bitmaps have diameter D, and in the first bitmap the pattern is located to the left of the pattern in the second bitmap, then we require that the first bitmap appear before the second in the enumeration. Thus, if in the bitmap whose index we are evaluating, the pattern is shifted s places (where $s=0$ if the pattern is at its leftmost location), then we add $s\times\binom{D-2}{k-2}$ to the previous sum to determine the starting index of patterns of diameter D at that location in the set.

But we can now continue recursively: the final enumeration value of the bitmap is the sum of the previous two terms plus the enumeration value of a new pattern against a new background bitmap. The new background is defined as the substring between but not including the first and last 1-bits of the previous pattern; the new pattern is made up of the 1-bits and 0-bits between and including the extreme 1-bits of the remaining bits; and the shift value is defined as above in terms of the new pattern and background. It is the position of the new pattern in its enumeration, that is, the enumeration value of the remaining $k-2$ members within the set of $D-2$ elements, that is added to the previous sum. The process stops once the pattern has zero or one 1-bits.

The following example should clarify these concepts. Suppose $N=12$ and the initial bitmap with 5 bits turned on is represented by the bit-string 010011010100. The original pattern is thus 100110101, and is shifted 1 place. In the next step of recursion, the new bitmap is 0011010, for which the new pattern is 1101, shifted 2 places against a background of seven bits. In the next and final recursion step, the background bitmap has only two positions, and the pattern has only one 1-bit, which is shifted by zero places.

3.2 Simplifications

The above sums can be simplified using standard combinatorial relations. We first decompose the first sum into two pieces:

$$\sum_{d<D}\binom{d-2}{k-2}(N-d+1)=N\sum_{d<D}\binom{d-2}{k-2}-\sum_{d<D}\binom{d-2}{k-2}(d-1).$$

But in the second component we can replace $\binom{d-2}{k-2}(d-1)$ by $\binom{d-1}{k-1}(k-1)$ and take the $k-1$ outside of the summation. Both sums are now of the form: $\sum_{s<S}\binom{s}{n}$, yielding $\binom{S}{n+1}$. Thus our sum becomes: $N\binom{D-2}{k-1}-(k-1)\binom{D-1}{k}$, and rewriting the second binomial coefficient, $\left(N-\frac{(k-1)(D-1)}{k}\right)\binom{D-2}{k-1}$.

We can now formalize the enumeration. First we represent the bitmap as follows: $S = \big((N_1, k_1, d_1, s_1), (N_2, k_2, d_2, s_2), \ldots, (N_n, k_n, d_n, s_n)\big)$, where $N_1 = N$, the size of the bitmap and $N_i = d_{i-1} - 2$; d_i is the diameter of the i-th pattern within the N_i potential bit positions; and s_i is the amount the i-th pattern is shifted within its background. If $k_n = 1$ then $d_n = 1$ as well. The encoding begins with $k_1 = k$ 1-bits and at each stage the number of 1-bits is reduced by 2, so $k_i = k_{i-1} - 2$. For the above example this yields $S = (\ (12, 5, 9, 1), (7, 3, 4, 2), (2, 1, 1, 0)\ )$. It is easy to see how this sequence can be used to reconstruct the original bit-vector.

In terms of this representation, the value assigned to S is

$$1 + \sum_{1\le i\le n} \left(N_i - \frac{(k_i-1)(d_i-1)}{k_i}\right)\binom{d_i-2}{k_i-1} + \sum_{1\le i\le n} s_i \binom{d_i-2}{k_i-2}.$$

In the above formulæ, we define $\binom{n}{0} = 1$ if $n \ge 0$; $\binom{-1}{0} = 0$ and $\binom{-1}{-1} = 1$ (which is not the standard extension of the binomial coefficients, but is useful for our application; see Appendix for a detailed discussion of this definition). As usual, $\binom{n}{m} = 0$ if $n < m$, and $\binom{0}{m} = 0$ for $m > 0$.

As an example, let us evaluate the index of the bit-string $100\cdots01$, which corresponds to $S = (\ (N, 2, N, 0),\ (N-2, 0, 0, 0)\ )$. This is the bitmap with two 1-bits of largest diameter, and we expect it to have the largest possible index, $\binom{N}{2}$. The value of the index is indeed

$$1 + \left(N - \tfrac{(N-1)}{2}\right)\binom{N-2}{1} \;=\; 1 + \tfrac{N^2-N-2}{2} \;=\; \tfrac{N^2-N}{2} \;=\; \binom{N}{2}.$$

3.3 Implementation

For encoding, the formulæ could be used directly or via tables. In many applications, encoding will be done once, but the reconstruction of the set will be done often and must be reasonably efficient. For this reason it is worth examining this component of the implementation a little more closely.

We begin, then, knowing N and k and the index value I of the bit-string within the enumeration. Since if the first diameter is d_1, all bitmaps with smaller diameters appear earlier, and all with larger diameters appear later; thus, we can find the diameter of our bitmap by finding the largest value of d for which $\sum\left(N - \frac{(k-1)(d-1)}{k}\right)\binom{d-2}{k-1}$ is still smaller than I. Since the values of $\binom{d-2}{k-1}$ can be tabulated within a N^2 table (actually only half that size if zero values are suppressed), this comparison can be done quickly. If further speed improvement is required, two tables can be used, one as above and the other with values of $\frac{(k-1)(d-1)}{k}\binom{d-2}{k-1}$: this reduces the amount of calculation needed to compute the factor of the binomial coefficients. In any case, the storage overhead for these decoding tables is smaller than that required by standard enumerations, so that the new method is preferable in an application to data compression.

Once d is known, we can procede in a similar way, and with the same table, to compute the shift. Thus the positions of the first and last 1-bits of our bitmap are known. If we subtract the sum associated with these two steps from the index value of the bitmap, this reduces the problem to the case of a bitmap of $k-2$ from $d-2$ members, with the reduced index value. We continue until zero or one element remains, for which the result is immediate.

4. Choosing the encoding method

Since we have to generate many different codes, the Shannon-Fano method (as defined in [11]) seems the most appropriate if we are concerned with processing speed. Thus an element, which according to the model at hand appears with probability p, will be encoded by $\lceil -\log_2 p \rceil$ bits. Once the length of the codeword is determined, the actual codeword is easily generated. But Shannon-Fano codes are not optimal and might in fact be quite wasteful, especially for the very low probabilities.

While Shannon-Fano coding is fast, when high precision is required Huffman codes are a good alternative. Under the constraint that every codeword consists of an integral number of bits, they are optimal; however their computation is much more involved than that of Shannon-Fano codes, because here every codeword depends on the whole set of probabilities. Thus more processing time is needed, but compression is improved. On the other hand, Huffman codes are not effective in the presence of very high probabilities. Elements occurring with high probability have low information content, yet their Huffman codeword cannot be shorter than one bit. If this is a prominent feature, arithmetic coding must be considered.

Arithmetic coding more directly uses the probabilities derived from the model described above, and overcomes the problem of high probability elements by encoding entire messages, not just codewords. Effectively, an element with probability p is encoded by exactly $-\log_2 p$ bits, which is the information theoretic minimum. While in many contexts arithmetic codes might not improve much on Huffman codes, their superiority here might be substantial, because the model may generate many high probabilities. There is of course a time/space tradeoff, as the computation of arithmetic codes is generally more expensive than that of Huffman codes. On the other hand, as can be seen from our experimental results, arithmetic codes may perform worse than Huffman codes when encoding is based on estimated probabilities, as in our case, and when these estimates are not precisely describing the true values.

For arithmetic coding we have to enumerate the bit patterns and assign a cumulative probability to each element. We could e.g., use the enumeration developed in Section 3.

If k of the N bits are on, then the cumulative probability would be

$$\left[\sum_{r=0}^{k-1} \binom{N}{r} P^r (1-P)^{N-r}\right] + \Delta P^k (1-P)^{N-k},$$

where P is the probability of a bit being turned on and is computed as described in the next section; Δ is the index of the given N-bit pattern with k bits on, in the enumeration of all the N-bit patterns with k bits on, so $\Delta \le \binom{N}{k}$. This approach is direct and accurate. The Huffman encoding procedure encourages a method that is less exact, but perhaps more intuitive.

4.1 Estimating the model parameters

Given the bitmap table, we would like to compute the probability of any given bit pattern for each N-bit block. To do this we must first use our model to estimate the probability P_{ij} that the bit in row i and column j is set to 1. Using $P = P_{ij}$, i.e., assuming that the probabilities are constant within a block, the probability that k bits have been turned on, irrespective of which of the N bits in the block they are, is given by the binomial distribution, $\Pr(k \text{ bits on}) = \binom{N}{k} P^k (1-P)^{N-k}$.

We noted above that run lengths can usefully be taken into account. The modification of probabilities is immediate. Our primary set of elements to encode is now $N, N-1, \ldots, 1, 0, 0^2, \ldots, 0^m$, where 0^i denotes the run of i 0-blocks, so that runs of length up to m are represented. The probabilities for $N, \ldots, 1$ are given by the binomial distribution as before. But if we denote $(1-P)^N$ by Z, then $\Pr(0)$ is now given by $Z(1-Z)$, the probability of a zero block followed by a non-zero block. Similarly, the probability of 0^i for $i < m$ is given by $Z^i(1-Z)$, and $\Pr(0^m)$ is Z^m. It is easy to see that these terms sum to one. This however assumes the probability P is constant over the length of the run. When this cannot be reasonably assumed, we would use a probability P_j for column-block j: denoting $(1-P_j)^N$ by Z_j, the probability of 0^i_j, a run of i zero-blocks starting in column-block j is then

$$\Pr(0^i_j) = (1 - Z_{j+i}) \prod_{r=0}^{i-1} Z_{j+r} \quad \text{for } i < m, \qquad \text{and} \quad \Pr(0^m_j) = \prod_{r=0}^{m-1} Z_{j+r}.$$

If m is large enough to include all runs of 0, we must modify the probabilities of the blocks following a run of zero-blocks, to indicate that a zero-block is impossible.

The next idea is to reorganize of the entire bitmap. The original ordering of columns is induced by the nature of the database at hand. For a column corresponding to a document, the order is usually either chronological or classified by author or subject. The rows are mostly stored in the lexicographic order of the corresponding words. There is, however, little or no interaction between the number of different words in a document (that is, the number of 1-bits in a column) and the time the document has been produced; a similar remark is true for the rows. It follows that clusters of 1-bits of various sizes appear anywhere in the bit table, so that assigning a single probability to a block of rows or columns may result in biased estimates.

We therefore reorganize the rows and columns of the table so as to get monotonically non-increasing 1-bit density in both dimensions. Now considering the probability that a 1-bit is constant within an N-bit block will imply a smaller error. Thus relatively few parameters define the model. If further economy is desired, we could estimate the row and column probabilities by means of a bit generation model, thereby reducing the model to two or three parameters for the rows and two or three for the columns. This possibility is discussed in the following section. For decompression we have of course to remember the two inverse permutations, but since these are constant, the space overhead of reorganizing the table may be substantially smaller than the additional compression savings.

4.2 Bit generation models

4.2.1 Independent bit generation

Let us assume that we have an $R \times C$ table of bits, of which B have been turned on. We envision the table as having been generated as follows: beginning with all bits set to zero, each of the B 1-bits is randomly tossed onto the table; if it lands in a cell, the bit associated with the cell is turned on. Since we are dealing with sparse tables, we can, as a first approximation, ignore the possibility that two 1-bits will land in the same cell.

In defining the probabilities governing the random process described above, we must distinguish between the probability p_{ij} that, if one more 1-bit were generated, it would go to cell ij; and the probability, P_{ij}, that after all B bits have been generated, at least one has landed in cell ij—that is, that the bit in cell ij has indeed been turned on. The latter will then be used to compute the probability of any block pattern.

Let us model p_{ij} first. In our independence model of bit generation, each row has a probability p_i associated with it and each column a probability q_j. Then the probability that the next 1-bit generated will go to cell ij is assumed, by the independence model, to be given by $p_{ij} = p_i \times q_j$. The probabilities p_i and q_j can be estimated by the row and column density of bits: if $n_{i\cdot}$ bits arrived in row i, and $n_{\cdot j}$ bits arrived in col j, then we estimate p_i by $\frac{n_{i\cdot}}{B}$ and p_j by $\frac{n_{\cdot j}}{B}$. Thus, at most only $R + C$ parameters need be stored and transmitted to the decompression module. As will be elaborated on below, this may be reduced further if we can model the variation in $n_{\cdot j}$ and $n_{i\cdot}$.

We can now estimate the probability that any given bit has in fact been turned on, since we know the probability of any 1-bit landing in the cell containing that bit, and we know how many bits were generated. Specifically, if B bits are generated, the probability that at least one of these land in cell ij is given by $P_{ij} = 1 - (1 - p_{ij})^B$. This, then, is the probability that the ij-th bit has been turned on. Since $n_{i\cdot}n_{\cdot j}/B$ is small and B is large, we can approximate this as $P_{ij} \approx 1 - \exp(-n_{i\cdot}n_{\cdot j}/B)$.

4.2.2 Independent generation for denser maps

The first model assumed that the p_{ij} were small enough so that we could ignore bits being turned on more than once. A more complex model must take this possibility into account, both when estimating the initial probabilities p_i and q_j, and computing the final probabilities on which compression is based.

Although the table has B bits turned on, let us assume that B' bits were initially sent to the table, the difference being accounted for by multiple landings. Thus, for any given bit, the probability of its landing in cell ij is $p_i q_j$ and the expected number of bits landing in that cell is $B' p_i q_j$. We may assume that the actual number of bits coming to cell ij is approximately Poisson governed. Then the probability that no 1-bit arrives at the cell is $\exp(-B' p_i q_j)$, and thus the probability that at least one 1-bit arrives is $1 - \exp(-B' p_i q_j)$. This is the probability that the bit in cell ij is actually on. Summing over the columns, we get as the expected number of bits on in row i, $\sum_{j=1}^{C}(1 - \exp(-B' p_i q_j))$, and summing over the rows, we get the expected number of bits on in column j, $\sum_{i=1}^{R}(1 - \exp(-B' p_i q_j))$. We wish to estimate the parameters B', p_i and q_j from the data. To do this we equate the expected number of 1-bits to the actual number of 1-bits: $\sum_{j=1}^{C}(1 - \exp(-B' p_i q_j)) = n_{i\cdot}$ and $\sum_{i=1}^{R}(1 - \exp(-B' p_i q_j)) = n_{\cdot j}$.

Such a set of equations can be solved iteratively. More simply, let us estimate q_j by $1/C$ when estimating p_i, and p_i by $1/R$ when estimating q_j. If we do so, the equations simplify greatly: $C(1 - \exp(-p_i B'/C)) = n_{i\cdot}$, or $p_i B' = -C \ln(1 - n_{i\cdot}/C)$, and similarly $q_j B' = -R \ln(1 - n_{\cdot j}/R)$. Since both the p_i and the q_j must sum to one we conclude:

$$
\begin{aligned}
B'_p &= -\sum_i C \ln\left(1 - \frac{n_{i\cdot}}{C}\right) & \qquad p_i &= -\frac{C}{B'_p} \ln\left(1 - \frac{n_{i\cdot}}{C}\right) \\
B'_q &= -\sum_j R \ln\left(1 - \frac{n_{\cdot j}}{R}\right) & \qquad q_j &= -\frac{R}{B'_q} \ln\left(1 - \frac{n_{\cdot j}}{R}\right).
\end{aligned}
$$

We take as our estimate for B' the average value $(B'_p + B'_q)/2$. (These values can be used as the initial estimates of the more precise iteration process.)

We now have estimated p_i, q_j and B'. Thus the probability that bit ij is on is $P_{ij} = 1 - \exp(-p_i q_j B')$ and we can procede as before, using the binomial distribution to establish the compressed representation of the block.

Note that if R and C are large and $n_{i\cdot}$ and $n_{\cdot j}$ are relatively small, B'_p and B'_q are both close to B, since

$$B'_p = -\sum_i \ln\left(1-\frac{n_{i\cdot}}{C}\right)^C \simeq -\sum_i \ln(e^{-n_{i\cdot}}) = \sum_i n_{i\cdot} = B = \sum_j n_{\cdot j} \simeq B'_q.$$

Similarly, we get $p_i \simeq n_{i\cdot}/B$ and $q_j \simeq n_{\cdot j}/B$, so that this model reduces to the simple model for sparse tables of Section 4.2.1.

4.2.3 Generalizing uniform column density

The assumption of bits being generated independently might not be realistic in real-life applications. In fact, there is strong evidence that bit-positions sometimes interact. Since rows correspond to words in the database and columns to documents, the occurrence of a word in a given document often implies its occurrence also in adjacent documents, which might treat the same topic; such a row-cluster effect has been exploited in hierarchical bit-vector compression [6]. Similarly, if two words are semantically related, they will tend to appear in more or less the same documents, which leads to the appearance of 1-bit clusters in certain columns; this is the idea behind the compression method in [3]. We therefore consider now the following model.

If the column densities were known to be uniform, we would clearly use P_i, the density of the 1-bits in row i, as our estimate for all P_{ij} in row i. Here $P_i = \frac{n_{i\cdot}}{C}$, is the number of 1-bits in row i divided by the length of the rows, which is the number of columns C. Now if the column densities are not uniform, we want to change the probabilities for each column in row i accordingly.

The density of column j is measured by $Q_j = \frac{n_{\cdot j}}{R}$, the number of 1-bits in column j divided by the length of the columns, which is the number of rows R, so that the relative weight of column j should be proportional to $Q_j / \sum_{r=1}^{C} Q_r = \frac{n_{\cdot j}}{B}$, where B is the total number of 1-bits in the table. But in the uniform case, the vector of probabilities associated with the bit-positions in row i would be $(P_i, P_i, \ldots, P_i)$, with the elements summing to MP_i. Similarly, in the non-uniform case, we want the elements of the probability vector to sum to MP_i, so we finally get as our estimate for P_{ij}:

$$P_{ij} \;=\; P_i \,\frac{Q_j}{\sum_r Q_r}\, M \;=\; \frac{n_{i\cdot} n_{\cdot j}}{B}.$$

Since in some extreme cases, this value for P_{ij} might be larger than 1, we would then take $P_{ij} = 1$, i.e., the model would assume that bit-position ij is occupied by a 1-bit. Note that if $n_{i\cdot} n_{\cdot j}/B$ is small, this value of P_{ij} is very close to the one derived in Section 2.4.1.

We noted however in our tests that many probabilities were not estimated accurately. While overestimating a probability only results in a shorter codeword (which ultimately yields a sub-optimal, but perhaps still close to optimal, code), significantly underestimating some probabilities may result in very long codewords, which can seriously affect the average codeword length if the true probabilities are much higher. The problem is particularly severe for arithmetic and Shannon-Fano coding, where the codeword length is based only on the estimated probabilities. In contrast, the codeword lengths for Huffman codes depend also on the number of codewords, and minor changes in the probabilities do generally not change the code. In [14] an upper bound is given on a "pseudo-distance" between two probability distributions which still yield the same Huffman tree.

The possibility of having underestimated the probabilities in certain columns leads to a model in which the weight for column j is chosen proportionally to $\sqrt{Q_j}$, rather than to Q_j, or more generally to $Q_j^{1/k}$, for $k = 2, 3, \ldots$.

4.2.4 Linear regression models

For our next model, we assume that the density changes regularly over rows and columns, and accept the possibility that there may be an interaction between rows and columns. We now directly estimate the expected number of bits in a block by means of a regression equation. That is, suppose that in our actual table, there are n_{ij} bits turned on in block ij. Then we solve the linear regression equation:

$$n_{ij} = b_0 + b_1\, i \;+\; b_2\, j \;+\; b_{12}\,(i \;\times\; j).$$

We next use the parameters b_0, b_1, b_2 and b_{12} just evaluated to compute the expected number of bits turned on in block ij: if we expect E of the N bits in a block to be turned on, then we use E/N to estimate the binomial probability parameter, and the binomial distribution to estimate the actual probabilities. Note that the decoder needs only the regression coefficients and the parameters defining the table to reproduce this process, i.e., only 4 numbers have to be stored. Since n_{ij} is a non-increasing function in both dimensions, we expect b_1 and b_2 to be negative. However, the rate of decrease is not necessarily linear, which leads to our next model, corresponding to hyperbolic decrease rate:

$$n_{ij} = b_0 + b_1\,\frac{1}{i} \;+\; b_2\,\frac{1}{j} \;+\; b_{12}\,\frac{1}{i \;\times\; j}.$$

5. Experimental results

The database we chose for testing some of the above methods is the Hebrew Bible, consisting of 305514 words, which are partitioned into 929 chapters. We chose each document to be one chapter, so the length of each bitmap was $C = 929$. Restricting ourselves to the words appearing in at least 20 chapters, we got $R = 1478$ bitmaps. The total number of 1-bits was $B = 95472$. The size N of a block of columns was chosen as 32 bits, a block of rows consisted of 16 maps and run-lengths were generated up to $m = 10$.

We produced a new bit-table in which both row densities and column densities decreased monotonically. For comparison, all the procedures were also applied to the non-permuted, original bitmaps. We have not yet implemented the improvement in the encoding based on the new enumeration method of Section 3, for which the $\binom{N}{k}$ indices are themselves encoded by some variable length code, but assumed that $\lceil \log_2 \binom{N}{k} \rceil$ bits are needed to encode any N-bit block with k 1-bits. In order to give a fair estimate of the compression savings, we do not compare the sizes of the compressed files to the size of the full bit-table of $RC = 1478 \times 929 = 1,373,062$ bits, but rather to the information theoretic lower bound of the compressed file which we get when all bit-positions are assumed independent. The latter is calculated as follows: the overall 1-bit probability is $p = B/RC = 0.06953$; thus the entropy per bit is

$$H = -p \log_2 p - (1-p) \log_2 (1-p) = 0.36418,$$

yielding a file size of $HRC = 500,035$ bits. This value could be approached very closely by using arithmetic coding on individual bits. The methods of the present work take row and column interaction into account, so that this lower bound can be improved upon. Table 1 summarizes the experimental results. The compression values are the improvement in percent relative to the benchmark HRC, that is $100 \times (1 -$ size of compressed file $/500035)$. A negative value thus indicates that the compressed file was larger than HRC.

The first block of two lines corresponds to the model of independent generation of Section 4.2.1 and the second block to the model of denser maps of Section 4.2.2. The third and fourth blocks correspond to the models of Section 4.2.3, with column weight proportional to Q_j and $\sqrt{Q_j}$ respectively. The last two blocks correspond to the regression models of Section 4.2.4, taking as parameters the row and column numbers, or their inverses, respectively.

	bit-table	*Shannon-Fano*	*Arithmetic*	*Huffman*
independent generation	permuted	5.63	11.32	15.49
$P_{ij} = 1 - \exp(-p_i q_j B)$	original	5.95	10.45	14.25
indep. on denser maps	permuted	5.68	11.37	15.64
$P_{ij} = 1 - \exp(-p_i q_j B')$	original	5.83	10.33	14.09
column weight	permuted	5.36	11.04	15.28
prop. to Q_j	original	5.77	10.27	13.97
column weight	permuted	7.70	13.49	16.33
prop. to $\sqrt{Q_j}$	original	5.70	10.11	13.62
linear regression on	permuted	-6.40	-1.30	-4.70
row and column	original	5.34	9.61	12.86
linear regression on	permuted	7.18	12.70	15.10
row and column inverses	original	5.13	9.28	12.77

Table 1: *Experimental results*

We note that permuting the maps indeed gave improved results, except for the first linear regression model (fifth block). It might be surprising that in our tests, Huffman coding consistently yields better compression than arithmetic coding (except again for the first regression model on the permuted maps). We explain this by the fact that we base the construction of the codes on estimated probabilities, and that these estimates are not very accurate, as mentioned earlier.

For the independent generation model on denser maps (second block), the estimated number of 1-bits B' is 100230 for the original maps and 102344 for the permuted ones, i.e., 5–7% higher than the actual number B. The results, however, are very close, with the second model performing better on the permuted maps, and the first model on the original ones.

When generalizing the model of the third and fourth block to have the weight in column j proportional to $Q_j^{1/k}$, for $k = 3, 4, 5, 6$, we get for the Huffman coding on the permuted maps the compression values 16.53, 16.563, 16.565 and 16.51 respectively. For arithmetic coding the optimum was obtained for $k = 4$ with 13.95.

It is interesting to note that the models based on regression gave results which are only slightly inferior to those obtained from the more complex models. The regression coefficients appear in Table 2. The first block corresponds to the regression on row, column and row $\times$ column, the second block to regression on their inverses. We see that for the first model, the coefficients of row, column and row $\times$ column are very close to zero, i.e., the best linear approximation was an almost constant function. For the second model, the coefficients seem more significant. Note also that the sum of squared residuals (last column, entitled *SSR*), which gives a quantitative measure of the goodness of the approximation, are negatively correlated to the compression savings: the better the approximation (lower *SSR*), the better the compression, for all three of Huffman, arithmetic and Shannon-Fano codings.

	bit-table	*intercept*	X_1	X_2	X_3	*SSR*
$(X_1, X_2, X_3) =$	permuted	6.585	-0.084	-0.087	0.001	14430
$(r, c, r \times c)$	original	3.412	-0.008	-0.700	0.000	4308
$(X_1, X_2, X_3) =$	permuted	0.884	21.45	0.202	7.138	3132
$(\frac{1}{r}, \frac{1}{c}, \frac{1}{r \times c})$	original	1.996	-0.247	1.255	0.102	4866

Table 2: *Linear regression coefficients*

Comparing the compression results with other methods that have been suggested: the hierarchical method of [19], with a block size of 6 bits, yields -7.61% and the improved version in [6] 0.30%. Jakobsson's [12] method yields 7.42%, and the method in [3] based on XORing bitmaps with similar structure gives -1.97%.

An interesting fact to note is that by using lower precision for the probabilities, we actually improved compression by arithmetic and Shannon-Fano coding! This is because of the existence of bit-blocks of very low probability in the models, which would have been assigned long codewords. However, by the very fact that these bit-blocks appear in the table, their probability is higher than predicted, so shorter codewords are appropriate. Specifically, the probability of no block is less than $\varepsilon_0 = \frac{1}{T}$, where $T = RC/m$ is the total number of blocks, so we used this value as threshold. When the model predicted a probability smaller than ε_0 for a certain block, we used $\lceil -\log_2 \varepsilon_0 \rceil$ as the corresponding codeword length for arithmetic and Shannon-Fano codes, which is consistent with the condition of unique decipherability. For Huffman codes, there was no problem, since the algorithm works even in the presence of probabilities equal to zero.

In summary, the model based techniques of this paper seem to yield a significant improvement.

6. Concluding remarks

The purpose of this paper is to push further the concept of basing compression on a detailed model of object generation—in this case, bits in a set of bitmaps. Our results are for an ideal case, which tests the limits of improvement possible with this approach. In practice several additional considerations must be taken into account. For example, we assumed that the row and column bitmaps are ordered. In some cases, these may be approximately ordered naturally; otherwise we have to store additional information to relate the original bitmaps to the ordered set (for our example, this incurs an additional cost only for the columns, since pointers to the rows must be stored in any case if arbitrary rows are to be retrieved).

We are also concerned about the cost of creating codes for each block. Most likely, an adaptive method for producing them would be used, since the probabilities change slowly and regularly over the blocks making up one row. Alternatively, a set of codes could be created preliminary to decoding, and the most appropriate used for each block. Since each block is defined by a single parameter, P, an optimal set of intervals could be constructed. It might seem that having to store these codes is costly in space, defeating the purpose of compression. But it must be kept in mind that the only information that must be stored with the data are the parameters defining the model, for example, the regression coefficients. All tables or trees can then be constructed given only this information. For situations where space is costly, but time and computing power less so, such a tradeoff may well be acceptable.

A final concern is the occurrence of probabilities near one for zero blocks in some regions of the table. This might yield innefficient Huffman codes. A possibility, within the Huffman framework, is to use variable length blocks as in [10], but with the block length being calculated from the model's parameters.

Appendix: Comment on binomial coefficients

Some readers may be struck by the deviation of our formula for $\binom{m}{k}$ from other, more customary ones when $k < 0$. But there is justification for this deviation beyond the fact that it is convenient for our derivation. To argue this, we review the argument supporting the most commonly used definition and indicate an alternative that also has good properties.

One common approach to developing the binomial coefficients is as follows [9], [13], [16]: we first recognize that the binomial coefficients arise usually out of combinatorial considerations. Thus combinatorial arguments can be used to define $\binom{m}{n}$ when $m, n > 0$ (integer values are always understood here). This yields the familiar formula $\binom{m}{n} = \frac{m(m-1)...(m-n+1)}{n(n-1)...1}$. But this formula makes sense even if $m \leq 0$, so we next extend the definition to this domain. Thus we have defined $\binom{m}{n}$ for all values of m, for $n > 0$. Now we notice that these values satisfy the Pascal Triangle rule, and by demanding that this appealing property be true generally, we are finally able to fill in the table for $n \leq 0$. This yields Table 3.

But the binomial coefficients have many other nice properties. One of the most heavily used is the symmetry property, $\binom{m}{n} = \binom{m}{m-n}$, though this unfortunately holds only for positive m. Another is its role in the binomial expansion: $(1+x)^m = \sum_n \binom{m}{n} x^n$; this sum typically is taken over $n \geq 0$, but can be extended to $n < 0$ if for such n the binomial coefficient is defined as zero. We will assume the sum to go over all integer values.

Table 3: $\binom{m}{n}$

m \ n	-4	-3	-2	-1	0	1	2	3	4	5	6
-4	0	0	0	0	1	-4	10	-20	35	-56	84
-3	0	0	0	0	1	-3	6	-10	15	-21	28
-2	0	0	0	0	1	-2	3	-4	5	-6	7
-1	0	0	0	0	1	-1	1	-1	1	-1	1
0	0	0	0	0	1	0	0	0	0	0	0
1	0	0	0	0	1	1	0	0	0	0	0
2	0	0	0	0	1	2	1	0	0	0	0
3	0	0	0	0	1	3	3	1	0	0	0
4	0	0	0	0	1	4	6	4	1	0	0

The last result is particularly appealing, since it offers a unifying view of the table: if we define the value of $\binom{m}{n}$ as above, we immediately get the whole table without having to build it in stages. For example, $(1+x)^0 = 1$, so $\binom{0}{0} = 1$, and for $n \neq 0$, $\binom{0}{n} = 0$. Similarly, $\binom{m}{n} = 0$ for all $n < 0$.

But defining the binomial coefficients in this ways leads to a striking alternative definition of the table, which we call $\binom{m}{n}^*$, since the expansion, taken over all n, is not unique. For consider the expansion

$$(1+x)^m = x^m\left(1+\frac{1}{x}\right)^m = x^m\left(\sum_n \binom{m}{n}\frac{1}{x^n}\right) = \sum_n \binom{m}{n} x^{m-n} = \sum_r \binom{m}{m-r} x^r,$$

where we use the conventional binomial expansion (over $\frac{1}{x}$) to get the second equality, and the substitution $r = m-n$ to get the last equality. If we now define $\binom{m}{n}^*$ by means of the equation $(1+x)^m = \sum_n \binom{m}{n}^* x^n$ in the second expansion, we get $\binom{m}{n}^* = \binom{m}{m-n}$, valid for both m and n positive, negative or zero. This relation simply expresses the symmetry property of the binomial coefficients for positive m, but defines a distinct entity for $m < 0$. This yields Table 4.

Table 4: $\binom{m}{n}^*$

$m \backslash n$	-6	-5	-4	-3	-2	-1	0	1	2	3	4
-4	10	-4	1	0	0	0	0	0	0	0	0
-3	-10	6	-3	1	0	0	0	0	0	0	0
-2	5	-4	3	-2	1	0	0	0	0	0	0
-1	-1	1	-1	1	-1	1	0	0	0	0	0
0	0	0	0	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	1	0	0	0
2	0	0	0	0	0	0	1	2	1	0	0
3	0	0	0	0	0	0	1	3	3	1	0
4	0	0	0	0	0	0	1	4	6	4	1

The dual character of these tables is made clear from the following more general relation: $(x+y)^m = \sum_n \binom{m}{n} x^n y^{m-n}$. If we set $x = 1$, we get one set of values; if we set $y = 1$ we get the dual set. They are also related by their convergence properties as Taylor series: the first expansion is well defined for $|x| < 1$, the second for $|x| > 1$.

A final observation on the relationship between the two coefficients may be helpful. Note that the values for $m, n \geq 0$ are determined by the vertical half-column of ones ($n = 0$, $m \geq 0$) and the positive diagonal of ones ($m = n \geq 0$) in conjunction with the Pascal Triangle rule. Continued application of this rule permits a further extension of the coefficients for $n < 0$, thereby defining the table for all n, if $m \geq 0$. To extend the table to values $m < 0$, we can continue the *vertical column of ones* into the domain $m < 0$ (that is, $\binom{m}{0} = 1$ for all m) and then use the Pascal Triangle rule to get $\binom{m}{n}$ and Table 3; or we can continue the *diagonal of ones* into this region (so $\binom{m}{m} = 1$ for all m) and use the Pascal Triangle rule to get $\binom{m}{n}^*$ and Table 4.

Several comments are in order:

1. Table 4 is consistent with the definitions used in this paper—thus our equations are really in terms of $\binom{m}{n}^*$.
2. There are several properties we would hate to lose in a redefinition of the binomial coefficients. These are

   (a) They should be the same as the original for the positive values of m, n with which we are very familiar. Equivalently, for the cases where this makes sense, we should get values that apply to combinatorial arguments.

   Then, the most heavily used and familiar relations should hold. For us, these include the following, which we append as additional requirements:

   (b) the symmetry property, $\binom{m}{n} = \binom{m}{m-n}$;

(c) the Pascal Triangle Rule; and,

(d) the binomial expansion.

For us, property (d) holds by definition. And since the tables are identical for positive m, n, property (a) holds as well. Checking the tables suggests that property (c) holds, and this is easily verified from the defining property. Indeed, any relationship implied by the generating function applies to the dual table.

Property (b) is particularly interesting, for our derivation suggests a new interpretation. We now assert the symmetry property as follows: $\binom{m}{n}^* = \binom{m}{m-n}$, that is, the $m-n$ value is the same as the n value of the dual table; we can describe this as the *weak-symmetry* property. Since the tables are identical for positive m, n, this identity is true in the *strong* form for these values. But in the form given here, it is true for all integers.

References

[1] **Apostolico A., Fraenkel A.S.,** Robust transmission of unbounded strings using Fibonacci representations, *IEEE Trans. on Inf. Th.* **IT–33** (1987) 238–245.

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

[3] **Bookstein A., Klein S.T.,** Optimal graphs for bit-vector compression, *Proc. 13-th ACM-SIGIR Conf.,* Brussels, Belgium (1990) 327–342.

[4] **Bookstein A., Klein S.T.,** Flexible compression for bitmap sets, *Proc. Data Compression Conference*, Snowbird, Utah (1991) 402–410.

[5] **Bookstein A., Klein S.T.,** Generative models for bitmap sets with compression applications, *Proc. 14-th ACM-SIGIR Conf.,* Chicago; ACM, Baltimore, MD (1991) 63–71.

[6] **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; ACM, Baltimore, MD (1986) 88–97.

[7] **Cover T.,** Enumerative Source Encoding, *IEEE Trans. on Inf. Th.* **IT–19** (1973) 73–77.

[8] **Elias P.,** Universal codeword sets and representations of integers, *IEEE Trans. on Inf. Th.* **IT–21** (1975) 194–203.

[9] **W. Feller,** *An Introduction to Probability Theory and its Applications, (2nd Ed.), Vol. I,* (Wiley, NY, 1950).

[10] **Fraenkel A.S., Klein S.T.**, Novel compression of sparse bit-strings, in *Combinatorial Algorithms on Words*, NATO ASI Series Vol **F12**, Springer Verlag, Berlin (1985) 169–183.

[11] **Hamming R.W.,** *Coding and Information Theory,* Prentice-Hall, Englewood Cliffs, NJ (1980).

[12] **Jakobsson M.**, Huffman coding in bit-vector compression, *Inf. Processing Letters* **7** (1978) 304–307.

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

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

[15] **Reingold E., Nievergelt J., Deo N.,** *Combinatorial Algorithms: Theory and Practice*, Prentice-Hall, Englewood Cliffs, NJ (1977).

[16] **J. Riordan,** *An Introduction to Combinatorial Analysis,* (Princeton University Press, Princeton, NJ, 1978).

[17] **Schuegraf E.J.**, Compression of large inverted files with hyperbolic term distribution, *Inf. Proc. and Management* **12** (1976) 377–384.

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

[19] **Wedekind H., Härder T.**, *Datenbanksysteme II*, B.-I. Wissenschaftsverlag, Mannheim (1976).

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