STREAMLINING CONTEXT MODELS FOR DATA COMPRESSION

Debra A. Lelewer and Daniel S. Hirschberg
Department of Information and Computer Science
University of California, Irvine, CA 92717

Abstract

Context modeling has emerged as the most promising new approach to compressing text. While context-modeling algorithms provide very good compression, they suffer from the disadvantages of being slow and requiring large amounts of main memory in which to execute. We describe a context-model-based algorithm that runs significantly faster, uses much less space, and provides compression ratios close to those of earlier context modeling algorithms. We achieve these improvements through the use of self-organizing lists.

Introduction

The most widely used data compression algorithms, including the Unix utility *compress*, are based on the work of Ziv and Lempel [ZL78]. These are dynamic algorithms that build a dictionary representative of the input text and code dictionary entries using fixed-length codewords. *Compress* typically reduces a file to approximately 50% of its original size and is extremely fast, but has a large memory requirement (450 Kbytes). Algorithm FG, an updated version of the Ziv-Lempel algorithm, requires less memory (186 Kbytes for encoding and 130 Kbytes for decoding) and achieves compression that is about 30% better than that provided by *compress* [FG89].

Newer approaches to data compression tend to focus on files of one particular type, and text files are most commonly studied. The most promising new methodology is one that predicts successive characters taking into account the context provided by characters already seen. What is meant by *predict* here is that previous characters are used in determining the number of bits used to encode the current character. A method of this type is referred to as a *context model* and, if the number of previous characters used to make a prediction is constant, an *order-i context model*. When $i = 0$, no context is used and the text is coded using unconditioned character counts (i.e., one character at a time). When $i = 1$, the previous character is used in encoding the current character; when $i = 2$, the previous two characters are used, and so on. A context model provides a frequency distribution for each context (each character in the order-1 case and each pair of characters

in the order-2 case). Arithmetic coding is used to map frequencies into code bits. Context modeling is generally used adaptively so that it compresses the data in a single pass. The state-of-the-art in context modeling is represented by Algorithm PPMC [BCW90].

A disadvantage of context modeling is that the memory requirement of the model frequently exceeds the size of the file being compressed. Bell et al. report average compression ratios (compressed file size divided by original file size expressed as a percentage) for PPMC of approximately 30% [BCW90]. PPMC uses 500 Kbytes of memory to represent a blended order-3 context model. While this quantity of memory may be available on research or production machines, it is not generally available. In particular, microcomputer implementations must greatly reduce memory utilization. Another disadvantage of context models is that they tend to be much slower than the Lempel-Ziv style of compression. Bell et al. report encoding and decoding speeds of 2000 characters per second (cps) for the order-3 context model as compared with 12000 cps for *compress* and 6000 cps for algorithm FG.

The algorithm we describe improves the practicality of the context modeling concept. Our modifications of the basic finite-context model improve its speed and decrease its memory requirements. Empirical experiments show our algorithm to be competitive with algorithms FG and PPMC. The advantage of algorithm FG is its speed; its compression effectiveness is less than that of algorithm PPMC. To achieve the superior compression performance of algorithm PPMC, speed is sacrificed and more than twice as much memory is required. Our method is closer in speed to algorithm FG, and with a space requirement of approximately 100 Kbytes (far less than both FG and PPMC) we achieve much of the improved compression evidenced by algorithm PPMC over algorithm FG. We present our algorithm from the point of view of the encoder, describing the way in which the encoder maintains a context model and uses corresponding frequency values to code characters. As in any adaptive data compression algorithm, the decoder must maintain the same model and use the frequency information in a compatible way so as to correctly interpret data received from the encoder.

Finite-Context Modeling

In this section we present an introduction to context modeling. We include only that information needed to understand our approach to the use of context. A more comprehensive discussion can be found in the book by Bell et al. [BCW90]. In the previous section, we defined an order-i context model to be one in which

the previous i characters are used to code the current character. Such a model should more accurately be referred to as a *pure* order-i model to distinguish it from the more common *blended* model. A blended model of order i is one in which the prediction of the order-i model is combined with predictions by models of lower orders (e.g., $i-1, i-2, \ldots, 0$) to form a final prediction. Blending is desirable and essentially unavoidable in an adaptive setting where the model is built from scratch as encoding proceeds. When the first character of a file is read, the model has no history on which to base predictions. Larger contexts become more meaningful as compression proceeds.

In a typical blended order-i model, the number of bits used to code character c will be dictated by the preceding i characters if c has occurred in this particular context before. Otherwise, models of lower order are consulted, typically beginning with order $i-1$, until one of them supplies a prediction. For each order-j model that fails to predict the current character, the encoder must emit an *escape* code, a signal to the decoder that a lower-order model is being consulted. The order-0 model may be initialized to provide a prediction for each character so that the process of consulting lower-order models terminates. Alternatively, the order-0 model may be used only for characters that have appeared before but are now appearing in a novel context. In this case, a model of order -1 (in which each character is equally likely) is used for predicting characters when they occur for the first time. When a character occurs in a novel context, this new information is added to the model being constructed.

We call an order-i context model *full* if for all $j \leq i$, every *j-gram* (sequence of j contiguous characters) that occurs in the file being encoded forms an order-j context in the model being constructed. A full model of even order 3 is rare since the space required to store context information for every 3-gram, 2-gram, and single character in the file is prohibitive. The PPMC algorithm of Bell et al. uses a full context model of order 3 stored in a tree data structure that is allowed to grow to 500 Kbytes [BCW90]. The model is rebuilt from scratch when it reaches this limit. We consider strategies that use less space, execute faster and achieve very close to the same compression performance. There are two earlier approaches to the problem of context modeling with modest memory requirements. Context-model-based algorithms by Abrahamson ([A89]) and Langdon and Rissanen ([LR83]) are described in [LH90]. These algorithms use order-1 context only and provide compression comparable to that of *compress*. Neither of these methods compresses

as effectively as algorithms FG and PPMC; nor do they generalize naturally to higher-order contexts.

In order to define a context-model-based algorithm, we need to determine maximum order and the type of blending strategy (if any) to be used. In addition, we must define data structures that represent the context model and the associated frequency information. Our approach differs from that of PPMC in both the method of blending and the choice of data structures.

Fast Order-3 Context Models in Limited Memory

In this section we describe an approach to context modeling in which the maximum order of the model, the amount of internal memory used, and the exact method of blending are parameters. In essence, the approach defines a family of algorithms where each algorithm in the family corresponds to a different set of parameter values. We have experimented extensively with models of maximum orders 2 and 3. In an earlier paper we describe experiments with order-2 modeling that include an order-2-and-0 model that achieves approximately 40% compression using only 48 Kbytes of internal storage [LH90]. In this paper we focus on the blended order-3 context model that gives the best overall results. These results include achieving 90% of the compression performance of algorithm PPMC using only 20% as much space.

The best algorithm in our family is based on an order-3-1-and-0 context model. That is, we construct a prediction for the character being encoded by blending predictions based on the previous three characters, the previous character, and unconditioned character counts. The order-3 and order-1 context information is maintained in the form of self-organizing lists and associated frequency values. The space requirements of the algorithm are determined by two parameters, $s1$ and $s3$, the sizes of the self-organizing lists for contexts of order 1 and order 3 respectively. For each trigram (i.e., context of order 3) appearing in the file we maintain a list of $s3$ successor characters and a corresponding frequency distribution. For each single character (context of order 1) we maintain a list of $s1$ successors and corresponding frequency information. The order-0 data consists of a frequency for each character.

Each character to be encoded may be said to occur in some three-character context; we arbitrarily select three predecessors for the first character of a file. We encode character z occurring in context wxy by event k if z occurs in position k of the list for context wxy. If z does not appear on wxy's list, we code an escape and

consult the list for the order-1 context y; if z occurs in position j of list y we code j; otherwise we emit another escape code. When neither context wxy nor context y predicts z we follow the two escape codes with an order-0 prediction (i.e., we code the character itself). If the list for context wxy (likewise context y) is empty, the corresponding escape code is not necessary. The decoder maintains the same model of the data as the encoder and knows that since context wxy has never occurred before it cannot supply a prediction.

Our prediction strategy is similar to that of algorithm PPMC except that PPMC uses orders 3, 2, 1, 0, and -1 while we use only models of orders 3, 1, and 0. Eliminating the models of order 2 and order -1 contributes to both the decreased memory requirement and increased speed of our method. Maintaining only $s3$ successors for each order-3 context (respectively $s1$ successors per order-1 context) also contributes to time and space savings. Limiting the number of predictions per context limits the space needed to store the model and the time needed to update it.

The update strategy we use is one that Bell et al. call *update exclusion* [BCW90]. That is, we update only those lists and frequency distributions that contribute to the prediction of the current character, z. If list wxy exists, we update it by either adding z (if it has never appeared in context wxy before) or transposing z with its predecessor on the list. If no wxy list exists then one will be created. If context wxy does not predict z and context y exists, then the y list is updated using the transpose heuristic. If the wxy context predicts z, context y is not used; if the y list exists it is not consulted and if it does not exist it will not be created. We also update each frequency distribution used in the prediction. When list wxy exists, the wxy frequency distribution is updated after it is used to encode either a list position or an escape. When context wxy does not predict and list y exists, the y frequency distribution is updated. The order-0 frequency distribution is updated whenever the character itself is coded.

An obvious disadvantage to fixing the sizes of the order-3 and order-1 context lists is that the lists are likely to be too short for some contexts and too long for others. When an order-3 list (say, list wxy) has $s3$ items and a new character z occurs in context wxy, we delete the bottom item (call it t) from the list and add z. Context wxy no longer predicts t. This does not affect the correctness of our algorithm; when t occurs again in context wxy it will be predicted by either context y or by the order-0 model. The fact that encoder and decoder maintain

identical models ensures correctness. In addition, the rationale behind the use of self-organizing lists is that we expect to have the $s3$ most common successors on the list at any point in time. As characteristics of the file change, successors that become common replace those that fall into disuse.

When an order-3 list contains fewer than $s3$ items we are subject to the criticism that we are not putting our memory resources where we need them. In fact, fixing the number of successors suggests the use of an array data structure rather than a linked structure; thus we avoid the space required for links and the time involved in creating and updating linked nodes. The links in a linked structure may also be viewed as consuming memory without directly representing information needed for prediction. Another advantage of the fixed-size structure is that it is not necessary to monitor its growth. In algorithm PPMC, the size of the trie is monitored until it reaches a limit at which time it is discarded and rebuilt. When rebuilding takes place, all of the information constructed from the prefix of the file is lost. By contrast, our model loses only the ability to predict certain successors in certain contexts, and only when they have ceased to occur frequently.

At this point it may not be clear that the memory requirements of our algorithm are modest. Even for small values of $s3$ and $s1$, storing a successor list for every order-3 context occurring in the file being compressed is very expensive. In order to conserve memory and have ready access to the lists, we store them in hash tables. The self-organizing lists for order-3 contexts are stored in a hash table to which every trigram can be mapped. The order-1 lists are stored in a table indexed by single characters. Since there are only n order-1 lists where n is the size of the character set, hashing is not needed here.

An obvious disadvantage to the use of hashing is the possibility of collision. If two or more contexts (say wxy and abc) hash to the same table position, the lists for these contexts are coalesced into a single self-organizing list used to represent both contexts. We can view this as wxy's successors vying with those of abc for position on the list. Intuitively, it would seem that our predictions are more accurate when wxy and abc are represented by separate lists. However, intuition is frequently unreliable when applied to the problem of compressing text. In fact, coalescing lists may actually improve compression performance. Hash conflicts have no impact on the correctness of the approach. We mitigate the negative effects of hashing in two ways. First, we select the hash function so as to minimize the occurrence of collisions. Second, we use linear probing to resolve collisions. We detect collisions by storing

with the self-organizing list an indication of the context to which it corresponds. We have found that a short linear probe is sufficient to provide good performance without a large cost in terms of speed.

Another way in which we conserve space is by limiting the number of frequency distributions more severely than the number of self-organizing lists. Our order-3 model is, then, essentially a two-level hashing scheme. A context hashes first to a position in the table of self-organizing lists and then to a smaller table of frequency distributions. Thus, even when contexts are not coalesced it is likely that frequency distributions are. The disadvantages of coalescing frequency distributions are essentially the same as those of coalescing lists. In addition to the parameters $s3$ and $s1$ that represent the lengths of the self-organizing lists for order-3 and order-1 contexts, respectively, we have additional parameters: $h3$, the size of the order-3 hash table for lists (i.e., number of order-3 lists stored); $f3$, the size of the order-3 hash table that stores frequency distributions (i.e., the number of order-3 distributions); and $f1$, the number of order-1 frequency distributions. The space requirements, speed, and compression performance of our method depend on the values of these parameters. The frequency information is stored in three tables, one for order 3, one for order 1 and one for order 0. For order 3 we have $f3$ sets of frequency information, each consisting of $s3$ frequency values, cumulative frequency values, and map values (assigning list positions to frequency values). The order-1 frequency data is stored in the same way. For order 0 we need only n frequency, cumulative frequency, and map values for the n symbols of our alphabet ($n = 256$ for 8-bit bytes).

Experimental Results

We compare the performance of our order-3-1-and-0 model to that of *compress*, algorithm FG and algorithm PPMC on a corpus of 14 files used by Bell et al. to measure the performance of a collection of data compression methods [BCW90]. The files represent a variety of sizes and types; *obj1* and *obj2* are executable files for two different machines, *geo* is a file of 32-bit numbers representing seismic data, *pic* is a bit map of a black and white facsimile picture. The remaining files are ASCII files of various types; *progc* is the source of *compress*, *progp* and *progl* are source files of LISP and Pascal programs, respectively.

In Table 1 we display compression performance achieved by algorithms FG, PPMC, and our order-3-1-and-0 model in terms of the number of bits per character in the compressed file. Original file sizes (in bytes) are given in column 2. The order-3-1-and-0 model used here has parameter settings: $s3 = 3$, $h3 = 12000$, $f3 = 900$,

$s1 = 25$, $f1 = 128$; and uses approximately 100 Kbytes of internal memory. The compression data for algorithms *compress*, FG and PPMC are taken from [BCW90].

File	Original Size	Order 3-1-0	Unix Compress	LZFG	PPMC
bib	111261	2.57	3.89	2.90	2.11
book1	768771	3.03	4.06	3.62	2.48
book2	610856	2.69	4.25	3.05	2.26
geo	102400	5.14	6.10	5.70	4.78
news	377109	3.22	4.90	3.44	2.65
obj1	21504	4.10	6.15	4.03	3.76
obj2	246814	3.33	5.19	2.96	2.69
paper1	53161	2.87	4.43	3.03	2.48
paper2	82199	2.85	3.98	3.16	2.45
pic	513216	0.90	0.99	0.87	1.09
progc	39611	2.87	4.41	2.89	2.49
progl	71646	2.06	3.57	1.97	1.90
progp	49379	2.12	3.92	1.90	1.84
trans	93965	2.02	3.94	1.76	1.77
Averages		2.84	4.26	2.95	2.48
Memory (Kbytes)		100	450	186	500

Table 1 Performance on selected files.

The parameter settings used to produce the results in Table 1 demonstrated the best overall compression given a limit on encode/decode memory of 100 Kbytes. Changing the parameter settings may provide improved compression on selected files. As an example, reducing the size of an order-3 list to 2 produced better results for files *geo* (5.05 bpc), *obj1* (4.08 bpc), and *pic* (.88 bpc). We have done some preliminary investigation into models of maximum order 4; an order-4-1-and-0 model provides improved compression for files *progl* (2.05 bpc), *progp* (2.11 bpc), and *trans* (1.93 bpc) using only 99 Kbytes of memory. An order-2-1-and-0 model

with a 99 Kbyte requirement provides much better performance on file *obj2* than any of our other limited-memory methods (3.13 bpc).

Future Research

Our algorithm provides an order-3 model that makes very efficient use of internal storage. The parameters of our model can be set at run time, so that it provides an unlimited variety of algorithms. We are attempting to develop guidelines for setting parameters depending on size and type of the file being compressed and on amount of internal memory available. We have not optimized our programs for speed. Preliminary experiments indicate that they are only slightly slower than the speeds reported by Bell et al. for algorithm FG [BCW90]. Our programs need to be tuned in order for speed comparisons to be really meaningful. We will also compare our results with state-of-the-art microcomputer compression programs. Our algorithms represent a major improvement over *compress*; it seems likely that our work represents an even more valuable improvement over limited-memory microcomputer versions of Lempel-Ziv coding. We are continuing to experiment with order-4 context modeling; our early experiments have produced limited success. None of our order-4 algorithms provide average compression that rivals that of the order-3-1-and-0 method, given approximately the same restrictions on the use of internal memory. It is possible that with a different selection of parameter values an order-4 model may provide improved performance.

Summary

We describe an order-3-1-and-0 finite context model that provides compression performance similar to that provided by algorithm PPMC using only 20% as much internal memory during encoding/decoding. Our method also runs faster than algorithm PPMC. We are able to achieve average compression factors of approximately 2.3 bits per character for source code files and approximately 2.8 bits per character for a large variety of file types using only 100 Kbytes of internal memory. In addition, we believe that our model is conceptually simple and easy to implement.

References

[A89] Abrahamson, D. M. An adaptive dependency source model for data compression. *Commun. ACM 32*, 1 (Jan., 1989), 77–83.

[BCW90] Bell, T., Cleary, J. G., and Witten, I. H. *Text Compression*, Prentice-Hall, Englewood Cliffs, N.J., 1990.

[FG89] Fiala, E. R. and Greene, D. H. Data compression with finite windows. *Commun. ACM 32*, 4 (Apr., 1989), 490–505.

[LR83] Langdon, G. G. and Rissanen, J. J. A double-adaptive file compression algorithm. *IEEE Trans. Comm. 31*, 11 (Nov., 1983), 1253–1255.

[LH90] Lelewer, D. A. and Hirschberg, D. S. An order-2 context model for data compression with reduced time and space requirements. Tech. Rep. 90-33. Dept. of Information and Computer Science, University of California, Irvine (Sept., 1990).

[ZL78] Ziv, J. and Lempel, A. Compression of individual sequences via variable-rate coding. *IEEE Trans. Inf. Theory 24*, 5 (Sept., 1978), 530–536.