Adaptive Weighing of Context Models for Lossless Data Compression

Matthew V. Mahoney

Florida Institute of Technology CS Dept.
150 W. University Blvd.
Melbourne FL 32901
mmahoney@cs.fit.edu
Technical Report CS-2005-16

Abstract

Until recently the state of the art in lossless data compression was prediction by partial match (PPM). A PPM model estimates the next-symbol probability distribution by combining statistics from the longest matching contiguous contexts in which each symbol value is found. We introduce a context mixing model which improves on PPM by allowing contexts which are arbitrary functions of the history. Each model independently estimates a probability and confidence that the next bit of data will be 0 or 1. Predictions are combined by weighted averaging. After a bit is arithmetic coded, the weights are adjusted along the cost gradient in weight space to favor the most accurate models. Context mixing compressors, as implemented by the open source PAQ project, are now top ranked on several independent benchmarks.

1. Introduction

The principle of Occam's Razor, applied to machine learning, states that one should choose the simplest hypothesis that fits the observed data. Hutter (2003), building on the work of Solomonoff (1986), formalized and proved this very general principle. Define an agent and an environment as a pair of interacting Turing machines. At each step, the agent sends a symbol to the environment, and the environment sends a symbol and also a reward signal to the agent. The goal of the agent is to maximize the accumulated reward. Hutter proved that the optimal behavior of the agent is to guess at each step that the most likely program controlling the environment is the shortest one consistent with the interaction observed so far.

Unfortunately, this does not solve the general machine learning problem. The algorithmic, or Kolmogorov, complexity of a string, defined as the length of the shortest program that outputs it, is not computable (Chaitin 1977). Thus, machine learning is an art: approximate solutions and heuristic methods are required..

Lossless data compression is equivalent to machine learning. In both cases, the fundamental problem is to estimate the probability $p(x)$ of event x (coded as a string) drawn from a random variable with an unknown (but presumably computable) probability distribution. Shannon and Weaver (1949) proved that the minimum expected code length of x given $p(x)$ is $\log_2 1/p(x)$ bits. Near-optimal codes (within one bit) are known and can be generated efficiently, for example Huffman codes (Huffman 1952) and arithmetic codes (Howard and Vitter 1992).

1.1. Text Compression and Natural Language Processing

An important subproblem of machine learning is natural language processing. Humans apply complex language rules and vast real-world knowledge to implicitly model natural language. For example, most English speaking people will recognize that $p($*recognize speech*$) > p($*reckon eyes peach*$)$. Unfortunately we do not know any algorithm that estimates these probabilities as accurately as a human. Shannon (1950) estimated that the entropy or information content of written English is about one bit per character (between 0.6 and 1.3) based on how well humans can predict successive characters in text. The best data compression programs achieve about 1.5 bits per character.

Language models can be used to improve the accuracy of speech recognition and language translation (Knight 1997), optical character recognition (Teahan et al. 1998), and for spell checking and solving substitution ciphers (Teahan and Cleary 1997). In each case, the problem is to find output string y that maximizes $p(y|x)$ for some input x. By Bayes law, this is equivalent to maximizing $p(x|y)p(y)$ where $p(x|y)$ is the inverse model (e.g. speech synthesis) and $p(y)$ is the language model.

Compression ratio can be used to evaluate a language model because it is minimized when the goal of matching the modeled distribution to the true distribution is met. However it is not the only function with this property. Chen, Beeferman, and Rosenfeld (1998) tested a number of proposed functions and did find that compression ratio (specifically, word perplexity) is among the best predictors of word error rate in speech recognition systems.

1.2. Online Modeling

Data compression ought to be a straightforward supervised classification problem. We are given a stream of symbols from an unknown (but presumably computable) source. The task is to predict the next symbol (so that the most likely symbols can be assigned the

shortest codes). The training set consists of all of the symbols already seen. This can be reduced to a classification problem in which each instance is the context, some function of the string of previously seen symbols (for example, a suffix of length n).

There are many well known techniques for solving such problems, for example, clustering, decision trees, neural networks, genetic algorithms, support vector machines, and so on. However, most data compressors do not use these techniques because they require offline training, i.e. multiple passes over the training set. A general purpose data compressor must be online. The input is arbitrary, so in general there cannot be any training data before the test data begins to arrive.

Until recently the best data compressors were based on PPM, prediction by partial match (Bell, Witten and Cleary 1989) with arithmetic coding of the symbols. In PPM, contexts consisting of suffixes of the history with lengths from 0 up to n (typically 5 to 8 bytes) are mapped to occurrence counts for each symbol in the alphabet. Symbols are assigned probabilities in proportion to their counts. If a count in the n-th order context is zero, then PPM falls back to lower order models until a nonzero probability can be assigned. PPM variants differ mainly in how much code space is reserved at each level for unseen symbols. The best programs use a variant of PPMZ (Bloom 1998) which estimates the "zero frequency" probability adaptively based on a small context.

One drawback of PPM is that contexts must be contiguous. For some data types such as images, the best predictor is the non-contiguous context of the surrounding pixels both horizontally and vertically. For audio it might be useful to discard the low order (noisy) bits of the previous samples from the context. For text, we might consider case-insensitive whole-word contexts. Unfortunately, PPM does not provide a mechanism for combining statistics from contexts which could be arbitrary functions of the history.

This is the problem we address in this paper. A *context mixing* algorithm combines the predictions of a large number of independent models by weighted averaging. The weights are adjusted online to favor the most accurate models. Compressors based on context mixing algorithms are now top ranked on most published benchmarks.

2. Context Mixing Models

A context mixing model works as follows. The input data is represented as a bit stream. For each bit, each model independently outputs two numbers, n_0, $n_1 \geq 0$, which can be thought of as measures of evidence that the next bit will be a 0 or 1, respectively. Taken together, it is an assertion by the model that the next bit will be 0 with probability n_0/n or 1 with probability n_1/n, where $n = n_0 + n_1$ is the model's relative confidence in this prediction.

Since models are independent, confidence is only meaningful when comparing two predictions by the same model, and not for comparing models. Instead we combine models by weighted summation of n_0 and n_1 over all of the models as follows:

$$
\begin{aligned}
S_0 &= \varepsilon + \Sigma_i\, w_i n_{0i} \quad \text{= evidence for 0} \\
S_1 &= \varepsilon + \Sigma_i\, w_i n_{1i} \quad \text{= evidence for 1} \\
S &= S_0 + S_1 \quad \text{= total evidence} \\
p_0 &= S_0/S \quad \text{= probability that next bit is 0} \\
p_1 &= S_0/S \quad \text{= probability that next bit is 1}
\end{aligned}
\tag{1}
$$

where $w_i \geq 0$ is the weight of the i'th model, n_{0i} and n_{1i} are the outputs n_0 and n_1 by the i-th model, and $\varepsilon > 0$ is a small constant to guarantee that $S_0, S_1 > 0$ and $0 < p_0, p_1 < 1$.

2.1. Updating the Mixing Weights

After coding each bit, the weights are adjusted along the cost gradient in weight space to favor the models that accurately predicted this bit. Let x be the bit just coded. The cost of optimally coding x is $\log_2 1/p_x$ bits. Taking the partial derivative of the cost with respect to each w_i in (1), with the restriction that weights cannot be negative, we obtain the following weight adjustment:

$$w_i \leftarrow \max[0,\, w_i + (x - p_1)(S n_{1i} - S_1 n_i) / S_0 S_1] \tag{2}$$

where $n_i = n_{0i} + n_{1i}$. The term $(x - p_1)$ is the prediction error.

Experimentally, equation (2) was found to be remarkably robust. The only tuning parameter is ε, and even this has very little effect on compression ratio over a wide range. Weights tend to grow at most logarithmically because the term $S_0 S_1$ in the denominator of (2) grows along with the weights. The weights can either be initialized to favor *a-priori* known better models, or simply be set to 0 to allow rapid initial training.

2.2. Arithmetic Coding

The predictions from equation (1) are arithmetic coded. The arithmetic code of a string x is the length of x together with a number in the half-open interval $[p_{<x},\, p_{<x} + p(x))$, where $p_{<x}$ is the probability that a string picked at random is lexicographically less than x. There is guaranteed to be a number in this interval with a base B encoding of not more than $1 + \log_B 1/p(x)$ digits (Howard and Vitter 1992).

When $p(x)$ is expressed as a product of conditional probabilities, $p(x_1 x_2 ... x_n) = \Pi_i\, p(x_i \mid x_1 x_2 ... x_{i-1})$ and the alphabet is binary, then the arithmetic code may be computed efficiently as follows. Begin with the range [0,1). For each bit x_i, divide the range into two parts in proportion to p_0 and p_1 from equation (1) and replace the range with the subrange corresponding to p_{xi}. In other words, if the range is [*low*, *high*) and the probability that x_i is 0 is p_0, then the range is updated as follows:

$$mid = low + p_0(high - low) \quad (3)$$
$$[low, high) \leftarrow [low, mid) \text{ if } x_i = 0$$
$$[mid, high) \text{ if } x_i = 1$$

As the range shrinks, the leading digits of the base B representations of *low* and *high* will match. These digits may be output immediately. It is convenient to use base B = 256 and represent each digit as a byte.

3. Modeling Nonstationary Data

Suppose that a certain context is observed n = 15 times and the next-bit sequence in this context is 000000000011111. What is the probability p_1 that the next bit will be a 1?

The answer depends on what we assume about the source. If we assume that the source is stationary (statistics do not change over time) and that the trials are independent (which seems unlikely from the given data) then we would count n_0 = 10 zeros and n_1 = 5 ones and guess $p_1 = n_1/(n_0 + n_1) = n_1/n$ = 1/3. We define a stationary update rule for a context model as follows:

Stationary Update Rule
Initialize $n_0 = n_1 = 0$.
If bit x is observed then increment n_x.

A simpler explanation (as demanded by Occam's Razor) might be that the source is not stationary, and that a state change occurred after 10 bits. We will model nonstationary sources as follows. We will predict that the last outcome will repeat, with confidence proportional to the number of consecutive repetitions. Essentially, we discard any bit counts that disagree with the most recent observation.

Nonstationary Update Rule
Initialize $n_0 = n_1 = 0$.
If bit x is observed then increment n_x and set $n_{1-x} = 0$.

In the example above, $n_0 = 0$ and $n_1 = 5$. We would predict $p_1 = 1$ with confidence 5.

In general, a source might or might not be stationary. For example, a document may contain pure text (stationary) or have embedded images (nonstationary). We define the following *semi-stationary* update rule as a compromise which empirically works well on a wide range of data. Rather than keep all counts, or discard all counts that disagree with the last observation, we discard about half of the counts. Specifically, we keep at least two and discard half of the rest, which has the effect of starting off as a stationary model and becoming more nonstationary as the counts grow.

Semi-stationary Update Rule
Initialize $n_0 = n_1 = 0$.
If bit x occurs then
  increment n_x
  if $n_{1-x} > 2$ then set $n_{1-x} = floor(n_{1-x} / 2 + 1)$.

For example, given the sequence 000000000011111 the state (n_0, n_1) would be updated as follows:

0000000000	(10, 0)	$p_1 = 0$, $n = 10$
00000000001	(6, 1)	$p_1 = 1/7$, $n = 7$
000000000011	(4, 2)	$p_1 = 1/3$, $n = 6$
0000000000111	(3, 3)	$p_1 = 1/2$, $n = 6$
00000000001111	(2, 4)	$p_1 = 2/3$, $n = 6$
000000000011111	(2, 5)	$p_1 = 5/7$, $n = 7$

4. Implementation: The PAQ Project

The PAQ series of context mixing compressors were developed as an open source project released under the GNU general public license. Source code is available at http://cs.fit.edu/~mmahoney/compression/

4.1. PAQ1

The first version, PAQ1, was developed in Jan. 2002 by M. Mahoney. It uses the following contexts:

- Eight contexts of length 0 to 7 bytes as general purpose models. All contexts also include the 0 to 7 bits of the current byte that precede the bit being predicted.
- Two word-oriented contexts of length 0 or 1 whole words preceding the currently predicted word (i.e. unigram and bigram models). A word is a case-insensitive sequence of the letters a-z.
- Two fixed-length record models for modeling two-dimensional data such as images and databases. One context is the column number and the other is the byte above. The record length is determined by detecting a series of 4 consecutive identical byte values with a uniform stride.
- One *match* context, which finds the last matching context of length 8 bytes or longer, and predicts whatever bit followed the match.

All models except *match* use semi-stationary update. A state (n_0,n_1) is represented as an 8 bit value using probabilistic updates to estimate large counts. Models are mixed as in equation (1) but the weights are fixed constants tuned empirically. The eight general purpose contexts of length n are weighted $w = (n + 1)^2$.

4.2. Secondary Symbol Estimation (PAQ2)

In May 2003 S. Osnach wrote PAQ1SSE (or PAQ2) which added SSE (secondary symbol estimation) after the mixer. SSE is a 2-D table which inputs the probability from the

mixer (quantized to 64 values non-uniformly with smaller steps near 0 and 1) and a small 10-bit context (the partially coded byte and match prediction) and outputs an improved probability. After the bit is coded, the SSE table entry is adjusted in proportion to the prediction error..

In Sept. 2003, M. Mahoney wrote PAQ3, which improved SSE by quantizing the input probability to 32 values with linear interpolation between adjacent table entries.

In Oct. 2003 S. Osnach wrote PAQ3N, adding three *sparse* models – two byte contexts that skip over intermediate bytes. Additional context was added to SSE.

4.3. Adaptive Model Mixing (PAQ4-PAQ6)

PAQ4 by M. Mahoney in Oct. 2003 introduced adaptive model weighting of 18 contexts as in Equation (2). The mixer uses 8 sets of weights, selected by a 3 bit context consisting of the 3 most significant bits of the previous whole byte.

PAQ5 in Dec. 2003 added a second mixer whose weights were selected by a 4-bit context consisting of the two most significant bits of the last two bytes. In addition, six new models were added for analog data (8 and 16 bit mono and stereo audio, 24-bit color images and 8-bit data). These contexts discard low order (noisy) bits.

PAQ6 in Dec. 2003 added nonstationary models (called run-length models) in addition to the semi-stationary updates for all models. A model was added for Intel executable code that translates relative CALL operands to absolute addresses in the context. There are also 10 general purpose contexts, 4 match models for long contexts, 5 record models, 9 sparse models, 7 analog models (including one for FAX images), and 6 word models including sparse bigrams.

In the first five months of 2004 there were 12 variations of PAQ6 by Berto Destasio, which included additional models, and changes to the semi-stationary update rule to discard counts more quickly and give greater weight to models in which one of the counts is 0. There were 7 versions by Fabio Buffoni, 2 by Johan De Bock, and 8 by Jason Schmidt, primarily optimizations for speed, and 3 by David A. Scott which improved the arithmetic coder. Eugene Shelwein and Jason Schmidt provided optimized compiles.

4.4. PAQAR

Between May and July 2004, Alexander Ratushnyak released 7 versions of PAQAR which greatly improved compression by vastly increasing the number of models at the expense of speed and memory. The latest version, PAQAR 4.0, uses 12 mixers with weights selected by different contexts. Each mixer has its own SSE stage. Those outputs are mixed by fixed weight averaging and passed through 5 more parallel SSE units whose outputs are again averaged. There are 10 general purpose contexts of length 0 to 9, 4 match models, 12 record contexts, 17 sparse contexts, 9 analog contexts, 13 word contexts, and 34 contexts for FAX images (for *pic* in the Calgary corpus). Models may be turned on or off when certain file types are detected. The Intel executable context model was replaced by a transform (filter) to change relative CALL and JMP operands to absolute addresses.

4.5. Dictionary Preprocessing (PAsQDa)

Between Jan. and July 2005, Przemyslaw Skibinski released 7 versions of PAsQDa which integrate a Word Reducing Transform (WRT) (Skibinski, Grabowski and Deorowicz 2005) into PAQ6 and PAQAR. WRT replaces English words with a 1 to 3 byte code from a dictionary. There are additional transforms to model punctuation, capitalization, and end of lines.

Malcolm Taylor added an independently implemented context mixing algorithm called PWCM (PAQ Weighted Context Mixing) to his commercial program, WinRK, in 2004, surpassing PAQAR and PAsQDa in many benchmarks.

5. Experimental Results

Table 1 shows compression results for major releases of PAQ and some popular and top ranked compressors (as of Oct. 2005) on the 14 file Calgary corpus, a widely used benchmark. The compression algorithm is shown in parenthesis with "+d" to indicate dictionary preprocessing. Options are selected for maximum compression. Files are compressed into a single archive if possible, which allows for modeling across files (solid mode). Compression times are in seconds on a 750 MHz Duron with 256 MB memory running Windows Me. Times marked with an asterisk are for programs that exceeded 256 MB and are estimated as 3.6 times the actual time (based on empirical timing comparisons) on an AMD 2800 with 1 GB memory.

Table 1 includes *compress*, *pkzip*, *gzip*, and *winrar* because of their popularity, *sbc* (S. Markinen, http://sbcarchiver.netfirms.com/) as the top ranked BWT compressor, and *slim* (S. Voskoboynikov, http://www.bars.lg.ua/slim/) and *durilca* (D. Shkarin, http://compression.ru/ds) as the top ranked PPM compressors without and with dictionary preprocessing, respectively. WinRK is the only compressor besides PAQ to use context mixing. The other compression algorithms are explained in the next section. Decompression times for LZW, LZ77 and BWT are considerably faster than compression, but are about the same for PPM and CM.

Table 1. Calgary corpus compression results

Program (type)	Options	Size	Time
Original size		3,141,622	
compress (LZW)		1,272,772	1.5
pkzip 2.04e (LZ77)		1,032,290	1.5
gzip (LZ77)	-9	1,017,624	2
winrar 3.20 b3 (PPM)	best	754,270	7
sbc 0.970r2 (BWT)	-b8 –m3	738,253	5.5
slim 0.021 (PPM)		658,494	156
durilca v.03a (PPM+d)	readme	647,028	35
paq1 (CM)		716,704	68
paq2 (CM) adds SSE		702,382	93
paq4 (CM) adds eq.(2)		672,134	222
paq6 (CM)	-6	648,892	635
paqar 4.0 (CM)	-6	604,254	2127*
pasqda 4.1 (CM+d)	-5	571,127	1586*
WinRK 2.0.1 (CM)	pwcm	617,240	1275
WinRK 2.0.1 (CM+d)	& dict	593,348	1107

5.1. Independent Benchmarks

Context mixing programs are top ranked by compression ratio (but not speed) in five actively maintained, independent benchmarks of general purpose lossless data compression programs. Results change rapidly but are current as of Nov. 14, 2005. All benchmarks below have been updated within the last month.

5.1.1. Calgary Challenge. The Calgary challenge (http://mailcom.com/challenge/) is a contest sponsored since 1996 by Leonid A. Broukhis with nominal prize money to compress the Calgary corpus. The size includes the decompression program (contained in a standard archive) in order to discourage dictionary preprocessing, which otherwise artificially inflates a ranking by moving information from the compressed data to the program. The current record of 596,314 bytes is held by A. Ratushnyak using a variant of PAsQDa with a tiny dictionary of about 200 words, set Oct. 25, 2005. The decompressor is semi-obfusticated C++ source code.

5.1.2. Maximum Compression Benchmark. (W. Bergmans, http://www.maximumcompression.com). This tests 142 compression programs on 10 files of various types totaling about 52 MB, with options set for best compression. The top ranked compressor for nine of the ten files were context mixing compressors (PAsQDa 4.1b or WinRK 2.0.6/pwcm).

5.1.3. UCLC Benchmark. (J. de Bock, http://uclc.info). This tests 92 programs on 8 data sets totaling about 172 MB. WinRK is top ranked on four and PAsQDa on two. The other two data sets, grayscale images and audio, are topped by specialized compressors.

5.1.4. Squeeze Chart Benchmark. (S. Busch (http://www.maximumcompression.com/benchmarks/Squeeze%20Chart%202005.pdf). This tests 156 programs (including versions) on 16 data sets totaling about 2.5 GB. Unlike the other benchmarks, the data was not released (except for the Calgary and Canterbury corpora), to discourage tuning compressors to the benchmarks. On this set, WinRK 2.1.6 was top ranked on six, PAQAR on four, PAsQDa on three, and *slim* on one.

5.1.5. EmilCont Benchmark. (B. Destasio (http://www.freewebs.com/emilcont/). This evaluates 393 compression programs (including different versions) on 12 unreleased files (text, images, audio, executable code) totaling 13 MB. By total size, the top 29 programs are WinRK and PAQ variants, followed by *slim*. Context mixing programs are top ranked on 10 of the 12 files.

6. Related Work

Lossless data compression originated in the 1830's with Morse code, which assigns the shortest codes to the letters occurring most frequently in English. Huffman (1952) devised an algorithm for assigning code lengths to symbols optimally given a probability distribution, although this code is not optimal in the sense of Shannon and Weaver (1949) because code lengths must be integers. Arithmetic coding (Howard and Vitter 1992) overcomes this limitation by assigning a code to the entire string.

The LZ family of codes (Ziv and Limpel 1978; Bell, Witten and Cleary 1989) are popular in spite of poor compression relative to PPM and context mixing because of their high speed, especially for decompression. The two main variants are LZ77, used in *gzip* and *zip*, and LZW used in GIF and UNIX *compress*. In LZ77, a repeated substring is replaced with a pointer to a previous occurrence. In LZW, a repeated substring is replaced with an index into a dictionary of previously seen substrings.

Compressors based on the Burrows-Wheeler transform (BWT) (Burrows and Wheeler 1994) are almost as fast as LZ and compress almost as well as PPM. A BWT compressor sorts the input characters by context and compresses the resulting stream with an adaptive order-0 model. A BWT model is equivalent to unbounded context length PPM (Cleary, Teahan, and Witten 1995).

PAQ1 is derived from an earlier compressor, P12, which predicts a bit stream using a 2 layer neural network with various contexts as inputs (Mahoney 2000). P12 is based on an earlier 3-layer neural network model which was trained offline by back propagation to predict text (Schmidhuber and Heil 1996). P12 and PAQ incorporated whole-word contexts based on an observation by Jiang and Jones (1992) that using whole words rather than letters as symbols improves text compression.

Model mixing and sparse word contexts are based on offline language models for speech recognition (Rosenfeld 1996). Order-2 and sparse word contexts were mixed using the maximum entropy approach, an iterative algorithm which converges to the most general distribution

that fits the training data. Kalai et al. (1999) describe online algorithms for combining language models by weighted averaging of probabilities. Weights are tuned online to favor better models. PAQ differs in that the submodels output a confidence in addition to a probability.

7. Conclusion and Future Work

In this paper we describe an online algorithm for combining arbitrary context models for binary strings that output both a prediction and a confidence. We also introduce a "semi-stationary" model which favors recent history over older evidence and expresses a high confidence in a prediction after a long string of all zeros or all ones. These techniques have been implemented in context mixing data compressors, which have now replaced PPM-based compressors at the top of the rankings of all of the major benchmarks.

Context mixing algorithms allow compressors to push the three way tradeoff between compression, speed, and memory to the extreme. The top ranked compressors, PAQAR, PAsQDa, and WinRK/pwcm, are 500 to 1000 times slower than popular compressors such as *gzip*, and require hundreds of megabytes of memory. The reason is the large number of models. Implementing context mixing algorithms efficiently remains a major challenge.

Acknowledgments

The PAQ project would not have been possible without dedicated volunteer work spanning four years by the developers mentioned in Section 4 and independent evaluators mentioned in Section 5. The sole source of funding for all involved was US$517.17 in winnings from the Calgary Challenge.

References

Bell, T.; Witten, I. H.; and Cleary, J. G. 1989. Modeling for Text Compression. *ACM Computing Surveys* 21(4):557-591.

Bloom, C. 1998. Solving the Problems of Context Modeling. http://www.cbloom.com/papers/ppmz.zip

Burrows M., and Wheeler, D. J. 1994. A Block-Sorting Lossless Data Compression Algorithm. SRC Research Report 124, Digital Systems Research Center.

Chaitin, G. J. 1977. Algorithmic Information Theory. *IBM Journal of Research and Development* 21:350-359, 496.

Chen, S. F., Beeferman, D., and Rosenfeld, R.. 1998. Evaluation Metrics for Language Models. In Proc. DARPA Broadcast News Transcription and Understanding Workshop.

Cleary, J. G.; Teahan, W. J., Witten, I. H. 1995. Unbounded Length Contexts for PPM. In Proc. Data Compression Conference:52-61.

Howard, P. G., and Vitter, J. S. 1992. Analysis of Arithmetic Coding for Data Compression. *Information Processing and Management* 28(6):749-763.

Huffman, D. 1952. A Method for the Construction of Minimum-Redundancy Codes. Proc. I.R.E.: 1098-1101.

Hutter, M. 2003. A Gentle Introduction to The Universal Algorithmic Agent {AIXI}. In *Artificial General Intelligence*, Goertzel B., and Pennachin C., eds., Springer.

Jiang, J. and Jones S. 1992. Word-based Dynamic Algorithms for Data Compression'', *IEE Proc. Communication, Speech, and Vision* 139(6):582-586.

Kalai, A..; Chen, S.; Blum, A.; and Rosenfeld R. 1999. On-line Algorithms for Combining Language Models. In IEEE Proc. Intl. Conf. on Acoustics Speech, and Signal Processing, 745-748.

Knight, K. 1997. Automatic Knowledge Acquisition for Machine Translation. *AI Magazine* 18(4):81-96.

Mahoney, M. 2000. Fast Text Compression with Neural Networks. In Proc. FLAIRS, Orlando FL.

Rosenfeld, R. 1996. A Maximum Entropy Approach to Adaptive Statistical Language Modeling. *Computer, Speech and Language* 10.

Schmidhuber, J. and Heil, S. 1996. Sequential Neural Text Compression. *IEEE Trans. on Neural Networks* 7(1):142-146.

Shannon, C., and Weaver, W. 1949. *The Mathematical Theory of Communication*. Urbana: University of Illinois Press.

Shannon, C. 1950. Prediction and Entropy of Printed English. *Bell Sys. Tech. J.* 3:50-64.

Skibinski, P.; Grabowski, S.; and Deorowicz, S. 2005. Revisiting Dictionary-Based Compression. To appear in *Software – Practice and Experience*. Wiley.

Solomonoff, R. 1986. The Application of Algorithmic Probability to Problems in Artificial Intelligence. in: M. Kochen and H. M. Hastings (Eds.), *Advances in Cognitive Science*, AAAS Selected Symposia Series, AAAS, Washington, D.C.:210-227, also in: L.N. Kanal and J.F. Lemmer (Eds.), *Uncertainty in Artificial Intelligence*, Elsevier Science Publishers B.V.:473-491.

Teahan, W. J., and Cleary, J. G. 1997. Models of English Text. *IEEE Proc. Data Compression Conference*:12-21.

Teahan, W. J.; Englis, S.; Cleary J. G.; and Holmes G. 1998. Correcting English Text using PPM Models. *IEEE Proc. Data Compression Conference*: 289-298.

Ziv, J. and Limpel, A. 1978. Compression of Individual Sequences via Variable-Rate Coding. *IEEE Trans. Information Theory* 24(5):530-536.