Protein is incompressible

Craig G. Nevill-Manning
Computer Science,
Rutgers University
Piscataway, NJ 08855
USA

Ian H. Witten
Computer Science,
University of Waikato,
Hamilton,
New Zealand

Life is based on two polymers, DNA and protein, whose properties can be described in a simple text file. It is natural to expect that standard text compression techniques would work on biological sequences as they do on English text. But biological sequences have a fundamentally different structure from linguistic ones, and standard compression schemes exhibit disappointing performance on them. We describe a new approach to compression that takes account of the underlying biochemical principles. This gives rise to a generalization of blending for statistical compressors where every context is used, weighted by its similarity to the current context. Results support what research in bioinformatics has shown—that there is little Markov dependency in protein. This cripples data compression schemes and reduces them to order zero models.

It is a surprising but convenient fact that life is based on unbranched polymers and can thus be represented as a linear sequence. Both DNA, which transmits data from one generation to the next, and protein, the machinery of life, can be transcribed into a simple text file. Text files beg to be compressed, especially ones like genomes that run to gigabytes, and it is not surprising that there have been many attempts to compress biological sequence data.

There are two quite different motivations for compression. First, from a practical perspective, compression enables efficient use of resources such as storage and bandwidth. Second, from a scientific perspective, it provides a way of capturing and quantifying structure in a sequence. Our interest is in this second aim: we seek to discover structure that corresponds to interesting biological insight. In a similar way, SEQUITUR (Nevill-Manning and Witten, 1997) captures hierarchical repetitive structure as a by-product of performing compression, and text mining approaches distil out structured information by using different compression models (Witten *et al.*, 1998).

The two motivations call for different evaluation techniques. The redundancy found in typical disk files containing protein structures, which provides a practical incentive for compression, may reflect sampling artifacts rather than anything fundamental about the structures themselves. Indeed, most available databases of protein sequences do not suit our purpose. They contain a sample of proteins from various organisms sequenced so far. Because these organisms are sometimes close in evolutionary distance, their proteins are very similar. This results in a high degree of redundancy in the databases, and significant compression can be achieved using ordinary general-purpose compression utilities. However, this compression is derived from the biased sampling of protein space used to produce the database, rather than anything intrinsic to proteins themselves. For this reason, we choose to compress the proteins from a single organism. Then, the selection of proteins is dictated by nature and cannot reflect any artificial bias. Any compression that can be contained should then reflect biologically meaningful structure in the sequences.

Standard compression techniques are unsuitable for protein structures, and very poor results are obtained using them. Textual substitution methods (such as LZ coding) exploit exact repetition, whereas protein sequences are subject to mutation that destroys repetition. Although probabilistic prediction methods (such as PPM) do not require repetition to be exact, they do assume a Markov process. Our experiments indicate that there is little empirical evidence for any significant degree of Markov dependency in protein.

In order to compress protein effectively, it is necessary to take advantage of the underlying biology. Proteins are strings constructed from an alphabet of amino acids. Unlike textual strings, a distance metric can be defined on the symbols of the alphabet. This distance reflects their mutation probabilities: symbols

are close together if they are likely to have been derived from the same symbol by mutation, and far apart otherwise. Such metrics have been studied by Cleary and Trigg (1998). A new form of blending is required which takes account of this distance to combine the predictions made by different contexts. Instead of predicting the next symbol based on several different context lengths, we sum over all possible contexts up to a certain length, weighted by their similarity to the current context. Under this scheme, there is no disadvantage, from a compression standpoint, to using a longer context than is required, because all contexts of a shorter length are implicitly included.

This paper develops these ideas. We begin by reviewing the structure of protein and its relation to DNA. Then we briefly review PPM, a standard compression scheme from which our new method has been developed. Next we describe the new scheme, CP for *Compress Protein.* Following that, we describe experiments used to evaluate its performance on three genomes. Then we review previous work on compressing biological sequences, and finally draw some conclusions.

Unfortunately, this work has produced a negative result. Although our scheme for compressing protein has encouraging properties, and performs much better than existing general-purpose compressors such as PPM and GZIP, it does not, in fact, yield structural insight into protein sequences. Indeed, the improvement in performance with increased model order is almost imperceptible. This leads to the conclusion that protein is incompressible, probably due to a combination of useful information and randomness from mutation. Whether incompressibility is an inevitable consequence of the evolutionary process or a fortuitous feature of the world we live in is a fascinating open question.

DNA vs protein

DNA is the genetic material of life—it transmits information from one generation to the next. Genes are regions within the genomic DNA that specify proteins. DNA can be represented as a sequence of symbols drawn from a four-symbol alphabet of nucleotides: A, C, G and T. To specify a protein, DNA is read a triplet, or codon, at a time, yielding $4^3 = 64$ different codons. Each codon represents one of the twenty amino acids, but the code is redundant: several different codons code for the same amino acid. The resulting sequence of amino acids constitute the protein specified by the DNA.

The genetic code is given in Table 1. It is interesting to note that abundant amino acids such as leucine are represented by as many as six codons: TTA, TTG, CTT, CTC, CTA and CTG. In the case of serine, leucine, proline, arginine, threonine, valine, alanine and glycine, the third symbol is irrelevant; one could construct a prefix-free code with shorter average code length. Reengineering the cell to interpret the more compact code would, however, be quite challenging.

	T	C	A	G	
T	F	S	Y	C	T
	F	S	Y	C	C
	L	S	*	*	A
	L	S	*	W	G
C	L	P	H	R	T
	L	P	H	R	C
	L	P	Q	R	A
	L	P	Q	R	G
A	I	T	N	S	T
	I	T	N	S	C
	I	T	K	R	A
	M	T	K	R	G
G	V	A	D	G	T
	V	A	D	G	C
	V	A	E	G	A
	V	A	E	G	G

Table 1 The genetic code. A codon consists of a nucleotide from the left column, one from the top row, and the third from the rightmost column. For example, CTC codes for L, and AAT codes for N. * represents the stop symbol.

Certain organisms and certain regions of genomes display a preference for nucleotides G and C over A and T. The redundancy in the genetic code allows these compositional biases without restricting the available amino acids. For example, in a G+C rich region, leucine would be preferentially represented as CTC and CTG rather than TTA, TTG, CTT and CTA.

The amount of an organism's genome involved in coding proteins varies considerably from species to species. In *E. coli*, almost all DNA is coding. At the other extreme, *Fritillaria*, a flowering plant, devotes only 0.02% of its genome to coding DNA. Of the non-coding regions, some is required for the regulation of genetic expression, but much of the rest consists of highly repetitive DNA whose function, if any, is still under investigation. The work described here concentrates on compressing the proteins encoded in the coding regions of DNA, rather than the non-coding regions, for this is the structurally interesting problem. That is why this paper is entitled *Protein is incompressible* rather than *DNA is incompressible*.

Given a protein compression scheme, we can produce a compression scheme for the DNA coding regions by first compressing the corresponding protein sequence, then sending disambiguation information to specify which of the possible triplets gave rise to each amino acid.

The structure of protein

Proteins make up much of the structure of the cell, and also act as molecular machines to perform the work of transporting molecules, replicating DNA, transmitting signals, expelling waste, and so on. Proteins are sequences drawn from a 20 symbol alphabet of amino acids, listed in Table 2.

Alphabet structure

Amino acids vary considerably in size, structure and affinity for water. It is this variety that allows proteins to take on a multitude of shapes and perform a wide range of functions, such as the porin protein in Figure 1. Despite their variation, there is some similarity between individual amino acids. Some, such as glycine and alanine are small. Another group, including isoleucine, leucine and valine, are uncharged, and thus hydrophobic—they prefer not to be exposed to water. Phenylalanine, tyrosine and tryptophan contain a hexagonal aromatic ring. In general, one can define a similarity metric between pairs of amino acids—high for a pair that shares several properties, low for those with very different properties. Table 2 shows some of the properties of each amino acid.

code	full name	description
A	alanine	small
C	cysteine	contains sulfur
D	aspartate	charged
E	glutamate	charged
F	phenylalanine	aromatic ring
G	glycine	tiny
H	histidine	has a ring
I	isoleucine	small, hydrophobic
K	lysine	charged
L	leucine	small, hydrophobic
M	methionine	contains sulfur
N	aspartamine	charged
P	proline	reconnects with backbone
Q	glutamine	charged
R	arginine	charged
S	serine	hydroxyl group
T	threonine	hydroxyl group
V	valine	small, hydrophobic
W	tryptophan	aromatic ring
Y.	tyrosine	aromatic ring

Table 2 The 20 amino acids that make up proteins

Similarity can be argued from a chemical perspective, but there is another important source of evidence for amino acid relationships. When the DNA encoding a protein mutates due to processes such as mis-copying or cosmic ray damage, the corresponding protein sequence may change. [1] If the new amino acid has very different properties to the original, the function of the protein may be disrupted. In many cases, this will cause the organism to die, or at least be reproductively impaired, which means that the mutation will not be passed on. If the organism survives, the new protein sequence will propagate to the organism's progeny. When sequences from different organisms are compared, certain substitutions occur more frequently than others—precisely those that substitute amino acids for similar amino acids. For example, the observed frequency of an I → L substitution is much higher than I → C, because substituting leucine for isoleucine does not radically change the chemistry of the protein, whereas substituting cysteine does.

Mutation probabilities

Table 3 shows the probability that an amino acid will mutate to another over a certain period of time (approximately 62 million years in this example). The probabilities are expressed as percentages, and the rows sum to 100%. For example, the probability of A remaining the same is 17%,and the probability of A mutating to M is 2%. High probabilities on the diagonal indicate amino acids that are highly conserved. Cysteine (32%) has distinctive chemical properties due to a sulfur atom at its tip. Glycine (34%) is the smallest amino acid by an order of magnitude. Proline (33%) is the only amino acid whose side chain reconnects to the protein backbone. On the other hand, isoleucine prefers to mutate to leucine or valine than remain the same, and methionine prefers to become isoleucine, leucine or valine.

The redundancy that we wish to detect may come from two sources. First, new genes arise through duplication. Mistakes made while copying DNA and other cellular processes sometimes lead to a gene being duplicated somewhere else in the genome. After a duplication event, the genes diverge over time due to mutation. A separate source of similar sequences is convergent evolution, where sequences start off different, but end up occurring more frequently in the genome because certain sequences are useful for particular functions. In both cases, an exact match is highly unlikely, but the distance according to the substitution matrix in Table 3 will be small.

Figure 1 The porin protein forms a cylindrical channel in the cell wall to allow small molecules to enter and leave. Image by Peitsch *et al.* (1995)

PPM compression

We begin with one of the best-performing lossless data compression schemes: PPM (Cleary and Witten, 1984). To compress a sequence of symbols drawn from an alphabet $\sum$, PPM forms, for each successive symbol in the sequence, a probability distribution $p(s)$, $s \in \sum$. Each symbol s is transmitted using a code of length $\log_2 p(s)$ bits. PPM proceeds from left to right along the sequence, and bases the distribution p on the symbols that have already been seen, i.e. those to the left of the symbol being predicted. For example, if the current input ends in *aba*, PPM examines previous occurrences of *aba* and tallies the symbols that appeared next. Suppose that *aba* appeared ten times previously, and that twice it was followed by *a*, once by *b*, five times by *c* and twice by *d*. Then the probabilities that PPM would assign to *a*, *b*, *c*, *d* are 0.2, 0.1, 0.5 and 0.2 respectively. If the next symbol is *d*, PPM transmits a code of length $-\log_2 0.2$ (≈ 2.3) bits. If *d* has never been seen before in the context of *abc*, PPM tries to use the shorter context *bc*, and informs the decoder. If *d* has not been seen in *this* context, PPM continues to shorten the context to *c*, then to nothing, at which point the marginal distribution of symbols is used. If this is the first appearance of *d* in the sequence, a uniform distribution on unseen symbols is used. The ability to vary the context length based on the available data is called 'blending.'

Blending is employed because data is sparse and the frequencies of different subsequences is non-uniform. If a large training sequence were always available and the source was ergodic, a fixed-length context would be acceptable—no blending would be required. However, this is not usually the case. If n samples per context were considered adequate for prediction, then for a fixed context length of c symbols, the training sequence would need to be at least $n|\Sigma|^c$ symbols long. For protein, which has an alphabet of twenty symbols, 1000 samples per context and a context length of 5 would require a training sequence 3 billion symbols long, or three times the size of the human genome, due to be completed in 2005. Furthermore, contexts are not uniformly distributed. In fact, the distribution is highly skewed, so that the most common 5-gram in protein, LLLLL, occurs 37,000 times more frequently than the least common one, WWWWW. PPM addresses this problem by using short contexts for rare combinations of symbols and long contexts for

	A	R	N	D	C	Q	E	G	H	I	L	K	M	F	P	S	T	W	Y	V
A	17	4	3	3	3	3	5	9	2	5	7	5	2	3	3	10	6	1	2	8
R	5	21	5	4	1	6	6	4	3	3	6	15	2	2	2	5	4	1	2	4
N	5	5	19	10	1	4	6	8	4	3	4	6	1	2	2	8	6	0	2	3
D	5	4	9	25	1	4	11	6	2	3	4	6	1	2	3	7	4	0	1	3
C	8	2	2	2	32	2	2	4	1	6	8	3	2	3	2	6	5	1	2	7
Q	6	8	5	5	1	12	12	4	3	3	5	10	2	2	3	6	5	1	2	4
E	6	6	5	11	1	8	17	4	3	3	4	9	1	2	3	6	4	1	2	4
G	11	3	5	5	1	2	4	34	2	3	4	5	1	2	2	7	4	1	2	3
H	5	6	7	4	1	5	6	4	22	3	5	6	2	4	2	5	3	1	7	3
I	5	2	2	2	2	2	2	2	1	16	19	3	4	5	2	3	5	1	2	20
L	6	3	2	2	2	2	2	3	1	14	23	3	6	7	2	3	4	1	3	12
K	7	12	5	5	1	6	8	5	2	3	5	16	2	2	3	6	5	1	2	4
M	6	3	2	2	2	3	3	3	2	11	21	4	9	5	2	4	4	1	2	10
F	4	2	2	2	1	1	2	3	2	8	14	2	3	24	1	3	3	2	11	7
P	7	3	3	4	1	3	5	5	2	3	5	5	1	2	33	6	5	0	2	4
S	12	4	6	6	2	4	6	8	2	3	5	6	2	2	3	12	9	1	2	5
T	8	4	5	4	2	3	5	5	2	6	7	5	2	3	3	11	14	1	2	8
W	4	3	2	2	1	2	3	4	2	4	7	3	2	9	1	3	3	33	9	4
V	5	3	3	2	1	2	3	3	6	5	8	4	2	16	2	4	3	3	19	6
Y	8	3	2	2	2	2	3	3	1	19	15	3	4	4	2	4	6	1	2	16

Table 3 Probability of amino acid substitutions in viable proteins. Probabilities are expressed as the percentage of cases where the amino acid labeling the row mutates to the amino acid labeling the column. $p(A \rightarrow D) = 3\%$, $p(A \rightarrow S) = 10\%$

common combinations.

The CP compression scheme

In language, exact repetition is common. The word ‘hedgehog’ is almost always spelt in exactly the same way: spellings such as ‘hetgehog’ and ‘hedjhog’ are considered unacceptable even though they may sound the same when spoken, and their meaning is unambiguous when written. In contrast, biological sequences undergo constant mutation, and as long as the new sequence has similar properties, the mutation will be accepted, that is, passed on to the next generation. Thus exact repetition is overlaid with mutation, which is modeled by the distance function. It is clearly desirable to take this into account in a compression scheme.

Instead of varying the context length based on the amount of data available, our compression scheme, CP, uses *all* contexts up to a certain length, weighted by their similarity to the current context. For a context length of one, the weight on a context is just the probability that the symbol in the context may have mutated to the current context. That is, if the current context is *I*, then, using the probabilities in the row for *I* in Table 3, *L* contributes 19% to the prediction, *H* contributes 1%, and *I* contributes 16%. In contrast, PPM would use 100% of the *I* context for the prediction. For longer contexts, the mutation probabilities are multiplied. If the current context is *II*, then *LL* contributes 20% * 20% = 3.8%, *II* contributes 16% * 16% = 2.5%, *HH* contributes 1%* 1% = 0.01%, and *HI* contributes 1% * 16% = 0.16%.

In general, the probability predicted for amino acid *aa* preceded by the context $s_0 s_1 \ldots s_l$ is given by

$$p_{s_0 s_1 \ldots s_l}(aa) = \sum_{c_0 \in A, c_1 \in A, \ldots c_l \in A,} f_{s_0 s_1 \ldots s_l}(aa) \prod_{k=0}^{l} d(s_k, c_k) \qquad (1)$$

where l is the context length, A is the alphabet of twenty amino acids, f are the observed frequencies of symbols in various contexts, and d is the probability of mutation given in Table 3.

The reasoning behind this weighting of contexts is that similar symbols occur in similar contexts, and by using all contexts, more data can be brought to bear on the problem of prediction. The statistics from all contexts of a particular length entail the statistics from all shorter contexts, because the distribution determined by a context is the normalized sum of all contexts for which it is a suffix. On a binary alphabet, for example, the probability distribution given by the context 01 is the sum of the distributions 001 and 101, normalized to (0,1), because every instance of 01 must have occurred in either the 001 or the 101 context. Similarly, the distribution following 01 is the normalized sum of all length 4 contexts ending in 01: 0001, 0101, 1001 and 1101.

Here is a more concrete example of how the summation technique replaces blending. Consider a protein sequence that ends in ILW, where ILW has not occurred before, but LW has. PPM would switch to the shorter context LW to make a prediction, even if LW occurs only once. In contrast, CP would use all length-three contexts, including LLW, VLW, ALW, LIW, LWW, and ALI. Because of the weighting, contexts such as LLW will contribute a great deal towards the final distribution, whereas ALI will contribute little. Note that all the contexts for which LW is a suffix, *LW, are included. Thus all of the frequency information in LW that PPM uses in its prediction is represented—though it is weighted differently.

Along with the context LW, CP also captures the context I*W and IL*. PPM’s emphasis on immediately preceding symbols is gone, because the distance calculation treats all positions the same. This difference makes sense because biological sequences have a fundamentally different nature from linguistic ones.

The function in equation (1) weights frequent contexts more highly. This has some merit because such contexts have less variance in the distributions that they produce. However, it may be better to weight contexts equally, in order to boost the contribution of similar contexts even if they have been seen less often. This is obtained by converting f to a proportion:

$$p_{s_0 s_1 \ldots s_l}(aa) = \sum_{c_0 \in A, c_1 \in A, \ldots c_l \in A,} \frac{f_{s_0 s_1 \ldots s_l}(aa)}{\sum_{a \in A} f_{s_0 s_1 \ldots s_l}(a)} \prod_{k=0}^{l} d(s_k, c_k) \qquad (2)$$

compression	HI (0.5 Mb)	SC (3 Mb)	MJ (0.45 Mb)	HS (3.3 Mb)
PPM	4.881	4.854	4.734	4.639
gzip	4.672	4.640	4.588	4.605
ORDER –1	4.322	4.322	4.322	4.322
ORDER 0	4.156	4.163	4.068	4.133
ORDER 1	4.149	4.158	4.060	4.126
ORDER 2	4.146	4.152	4.056	4.120
ORDER 3	4.143	4.146	4.051	4.112

Table 3 Compression rates in bits per character for four genomes: H. influenzae, S. cerevisiae, M. jannaschii, and H. sapiens.

In practice, there is almost no difference in the compression achieved using (1) and (2). The evaluation section below will present results based on (1).

Our approach is similar to Loewenstern and Yianilos' (1997) technique for compressing DNA. Instead of using mutation probabilities, they combine different context lengths with different Hamming distances from the current context. The Hamming distance is the discrete case of our approach—it is equivalent to using two mutation probabilities, instead of our 400: one probability is assigned to an exact match, and another probability to a mismatch. This is reasonable in DNA, where individual bases are less meaningful, and mutation probabilities are more uniform. They extend their technique to proteins as a way of compressing DNA and find that coding DNA, and hence the protein it codes for, is difficult to compress.

Evaluation

We used four genomes in our analysis. The first, Haemophilus influenzae (HI), is a bacterium that causes ear and respiratory infections in children. This genome was the first to be fully sequenced, and was made available in 1996. It is 1.83 megabases in size and contains approximately 1740 potential genes. When these genes are translated to proteins, the resulting file of amino acid sequences is 500Kb (representing each amino acid as one byte). The second genome, Saccharomyces cerevisiae (SC), or baker's yeast, has been studied as a model organism for several decades. At 13 megabases, it is the largest organism sequenced to date. The file of 8,220 protein sequences from S. cerevisiae is 2.9 Mb in size. The third genome, Methanococcus jannaschii (MJ), lives in very hot undersea vents and has a unique metabolism. It is 1.7 megabases in size, and has 1680 genes for a protein file size of 450 Kb. The final genome, Homo sapiens (HS), is incomplete: it includes 5733 human genes, for a file size of 3.3Mb.

Table 4 shows how various schemes compressed the files. The first, PPM, uses escape method D, a context of 7, and 255 megabytes of memory. The second scheme is gzip with the 'compress better' flag (–9). The remaining compression schemes are variants of CP. ORDER –1 encodes each amino acid in $\log_2 20 = 4.322$ bits, because it assumes a uniform, independent distribution of amino acids. ORDER 0 maintains the independence assumption, but takes the non-uniform distribution of amino acids into account. ORDER 1 sums over all contexts of length 1, weighted by the amino acid distance metric described above. ORDER 2 sums over all contexts of length 2, and ORDER 3 over all contexts of length 3. In our naïve implementation, adding one symbol of context increases running time by a factor of 20.

The simplest technique, ORDER –1, compresses at a baseline 4.322 bits per character (bpc). ORDER 0 improves this to 4.155, 4.163, 4.068 and 4.133 bpc for HI, SC, MJ and HS respectively. PPM performs comparatively poorly on all genomes, at 4.881, 4.854, 4.734 and 4.639 bpc—14.7% worse than ORDER 0 on average. Gzip fares better, with 4.672 4.640, 4.588 and 4.605 bpc, but is still 12% worse than ORDER 0 on average. ORDER 1 achieves 4.149, 4.158, 4.060 and 4.126, but the gains are negligible—0.2% on average. ORDER 2 averages 0.3% better on average, and ORDER 3 is 0.4% better. It appears that within a genome, little compression is possible using techniques that rely on Markov dependence.

Discussion

Despite the fact that we have found a way of getting materially better compression in this domain than is given by generic compression techniques, the result we have obtained is, unfortunately, a negative one. We have been unable to capitalize on prior context to improve compression over that achieved by a simple order-0 model—even using the domain-specific knowledge that is encoded in mutation probabilities. Unless there are other, more powerful, sources of background knowledge, the implication is that protein is incompressible. Its inherent complexity may simply be due to the random process of mutation that gives rise to it, or to the fact that it is already a highly compact representation of information. A combination of the two seems to be the most likely answer. In either case, attempts to use compression techniques to yield insight into the structure of protein are doomed to failure.

The technique used by CP to accommodate mutation probabilities may conceivably be applicable to other domains. It relies on a pre-specified, domain-specific, substitution matrix, and extending to another domain would require a similar matrix to be constructed. With English text, for example, although orthography does not permit random substitutions, similar letters can appear in similar contexts. A substitution matrix could be built by aligning subsequences of characters in text: *i* is close to *o* because they both occur in within *shop*/*ship*, *hot*/*hit*, and so on. Perhaps a better analogy to biological sequences is phonetic transcription. Common mistakes in transcription confuse voiced consonants with their unvoiced equivalents: *f* and *v*, *b* and *p*, *d* and *t*, and so on.

The basic idea of CP is to weight alternative contexts rather than choose the most likely one. A similar strategy is also used in compression schemes such as context tree weighting (Bunton, 1996), and in recently-developed theories of distance measurement (Cleary and Trigg, 1995). The idea of combining competing solutions, weighted by probability, instead of seeking the single best one, seems to be gaining momentum, not just in compression but in the bagging and boosting techniques that represent the current state of the art in machine learning (Breiman, 1996; Friedman *et al*., 1998). However, explicit, pre-determined, weighting probabilities that are characteristic of the protein domain are rarely available in other contexts.

References

Breiman L. (1996) “Bagging predictors,” *Machine Learning*, Vol. 24, No. 2, pp. 123-140..

Bunton, S. (1996) On-line stochastic processes in data compression. Ph.D. Thesis, University of Washington, Seattle, WA.

Cleary, J.G. and Trigg, L. (1995) “K*: An instance-based learner using an entropic distance measure.” *Proc Machine Learning Conference*, Tahoe City, CA, pp. 108-114.

Cleary, J.G. and Witten, I.H. (1984) “Data compression using adaptive coding and partial string matching.” *IEEE Trans Communications*, Vol. COM-32, No. 4, pp. 396-402.

Friedman, J., Hastie, T. and Tibshirani, R. (1998) “Additive logistic regression: a statistical view of boosting,” Technical Report, University of Toronto, Canada; July.

Henikoff, S. and Henikoff, J.G. (1992), “Amino acid substitution matrices from protein blocks,” *Proc Natl Acad Sci U S A* 1992 Nov 15;89(22):10915-9

Loewenstern, D. and Yianilos, P. (1997) “Significantly lower entropy estimates for natural DNA sequences,” *Proc. Data Compression Conference*, J.A. Storer and M. Cohn (Eds.), Los Alamitos, CA: IEEE Press, pp 151–160

Nevill-Manning, C.G. and Witten, I.H. (1997) “Compression and explanation using hierarchical grammars.” *Computer Journal*, Vol. 40, No. 2/3, pp. 103-116.

Peitsch, M.C., Stampf, D.R., Wells, T.N.C. Sussman, J.L. (1995) “The Swiss-3DImage collection and PDB-Browser on the World-Wide Web,” *Trends in Biochemical Sciences* 20:82-84.

Teahan, W.J., Inglis, S., Cleary, J.G. and Holmes, G. (1998) “Correcting English text using PPM models.” Proc Data Compression Conference, edited by J.A. Storer and J.H. Reif. IEEE Computer Society Press, Los Alamitos, CA, pp. 289-298.

Witten, I.H., Bray, Z., Mahoui, M. and Teahan, W.J. (1999) “Text mining: a new frontier for lossless compression.” Submitted to *Data Compression Conference*, Snowbird, Utah.