The PAQ1 Data Compression Program

Matthew V. Mahoney
Florida Tech., CS Dept.
mmahoney@cs.fit.edu
Draft, Jan. 20, 2002, revised Mar. 19.

Abstract

This paper describes the PAQ1 lossless data compression program. PAQ1 is an arithmetic encoder using a weighted average of five bit-level predictors. The five models are: (1) a bland model with 0 or 1 equally likely, (2) a set of order-1 through 8 nonstationary n-gram models, (3) a string matching model for n-grams longer than 8, (4) a nonstationary word unigram and bigram model for English text, and (5) a positional context model for data with fixed length records. Probabilities are weighted roughly by $n^2/tp(0)p(1)$ where n is the context length, t is the age of the training data (number of subsequent events), and p(0) and p(1) are the probabilities of a 0 or 1 (favoring long runs of zeros or ones). The aging of training statistics makes the model nonstationary, which gives excellent compression for mixed data types. PAQ1 compresses the concatenated Calgary corpus to 1.824 bits per character, which is 4.5% better than RK (Taylor, 1999) and 2.9% better than PPMONSTR (Shkarin, 2001), the top programs rated by Gilchrist (2001) and Ratushnyak (2001) respectively, although those programs do slightly better on homogeneous data.

1. Introduction

The best general purpose lossless compression algorithms are prediction by partial match, or PPM (Bell, Witten, and Cleary, 1989) and the Burrows-Wheeler transform, or BWT (Burrows and Wheeler, 1994). These are both stationary models, in the sense that the statistics used to estimate the probability (and therefore the code length) of a symbol are independent of the age of the training data. However, many types of data are nonstationary, in that newer statistics are more reliable. This can be shown by plotting the probability of matching two randomly chosen n-grams (sequences of *n* characters) as a function of *t*, the distance between them. Table 1 shows the result for Thomas Hardy's *Far from the Madding Crowd*, which is also *book1* from the Calgary corpus (Bell, Witten, and Cleary, 1989). Probabilities for the 768,771 byte file were estimated for each *t* by sampling one million random pairs separated by *t* plus or minus 10%.

n	t = 1	10	100	1000	10^4	10^5
1	2.133	6.450	6.500	6.427	6.411	6.403
2	0.027	0.650	0.734	0.701	0.691	0.689
3	0.016	0.129	0.193	0.166	0.161	0.157
4	0.013	0.045	0.078	0.063	0.056	0.056
5	0.012	0.017	0.037	0.025	0.021	0.021
6	0.011	0.010	0.017	0.011	0.007	0.006

Table 1. Percent probability of matching *n* consecutive characters separated by *t* in *Far from the Madding Crowd*.

The peak at around $t = 100$ is peculiar to natural language text, and is due to the tendency of words not to repeat in close proximity, as in *the the*. The long range drop off from 100 and 10^5 is due to the *cache* effect (Clarkson and Robinson, 1997), the tendency of words to be used in topic-specific contexts confined to a portion of the text. This effect can be used to improve compression by assigning higher probabilities (and shorter codes) to words that have already appeared recently.

The cache effect can be observed in all kinds of data, not just text. For example, the effect can be seen in the DNA sequence E.coli from the Canterbury corpus (Arnold and Bell, 1997) large file collection in Table 2. The file consists of 4,638,690 bytes from the alphabet *a, t, c,* and *g*.

n	t = 1	10	100	1000	10^4	10^5	10^6
1	26.263	25.840	25.481	25.183	25.025	25.035	24.989
2	6.760	6.978	6.739	6.541	6.429	6.429	6.355
3	1.903	1.971	1.832	1.719	1.692	1.663	1.677
4	0.566	0.564	0.513	0.461	0.453	0.438	0.441
5	0.146	0.164	0.151	0.124	0.124	0.118	0.116
6	0.030	0.047	0.045	0.034	0.032	0.032	0.033

Table 2. Percent probability of matching *n* consecutive DNA nucleotides separated by *t* in E.coli.

The cache effect can be found in 13 of 14 files in the Calgary corpus (all except *bib*) and in 13 of 14 files in the Canterbury corpus regular and large file sets (all except *kennedy.xls*). Yet neither of the best algorithms, PPM or BWT exploit this effect. For example, if the input to be compressed is

...can...can...can...cat...cat (1)

then to code the *t* at the end of the string, PPM would estimate its probability by counting characters in matching contexts (*ca*) with equal weight, regardless of their age. In this example, *t* occurred previously 1 out of 4 times in context *ca*, so that the next *t* would be assigned a code approximately $-\log_2 1/4 = 2$ bits long (neglecting for the moment the possibility of novel characters, such as *car*).

A BWT compressor sorts the input characters by their contexts prior to compression. For instance, context-sorting (1) might produce the sequence *tnntn* because they are all preceded by *ca*, and sorting brings them together. The exact order depends on the preceding characters (before the *ca*), and not on their original order, so the model is stationary regardless of the compression algorithm used on the transformed input.

Gilchrist's (2001) Archive Comparison Test (ACT) ranks 165 PC and Macintosh based compression programs (including different versions) on various test sets including the 18 file version of the Calgary corpus (adding four small text files, *paper3* through *paper6*). The top ranked programs as of July, 2001, from best to worst, are RK 1.02b5 (Taylor, 1999), RK 1.04 (Taylor, 2000), SBC 0.910 (Mäkinen, 2001), BOA 0.58b (Sutton, 1998), and ACB 2.00c (Buyanovsky, 1997). Ratushnyak (2001) ranked a smaller number of programs on the Canterbury corpus, and found the top three programs as of mid 2001 to be PPMONSTR var. H, PPMD var. H (Shkarin, 2001, 2002), and RK 1.02b5. All of these programs give 30-40% better compression than popular programs such as COMPRESS (1990), PKZIP (1993), or GZIP (Gailly, 1993), although they are slower and use more memory.

SBC is a BWT compressor. All of the others use PPM (except ACB, which uses associative coding). The PPM programs differ in how they compute the probability of novel characters. RK and BOA are based on PPMZ (Bloom, 1998). PPMZ uses an adaptive second level model to estimate the optimum value as a function of the order, the total character count, number of unique characters, and the last one or two bytes of context. RK also models certain contexts to improve compression on certain types of data including ASCII text, executables, and 8, 16, and 32 bit numeric data. RK and BOA both produces solid archives, using the statistics from one file to improve compression on another. They perform an analysis to determine the optimal ordering prior to compression.

PPMD and PPMONSTR are based on PPMII, or PPM with information inheritance (Shkarin 2002). PPMII, like PPMZ, uses a secondary escape model. Unlike PPMZ and most other PPM models, which uses statistics from the longest matching context, PPMII inherits the statistics of shorter contexts to set the initial estimate when a longer context is encountered for the first time. This has the effect of including statistics from contexts shorter than the longest match. PPMONSTR is a variation of PPMD that gives better compression at the cost of execution speed. The maximum order (longest possible context) is 16 for PPMD and 64 for PPMONSTR.

ACB uses associative coding. The input is sorted by context, as in BWT, but instead of compressing the transform, the context-sorted data is used to build a permuted index (but sorted reading backwards) which grows as data is read in. Suppose that the input string is *cx*, where *c* is the part already encoded (the context), and *x* is the remaining input. To encode *x*, we look up both *c* and *x* in the permuted index, matching as many characters as possible of *c* reading left and *x* reading right. Often, the best two matches are at the same place in the index, or separated by some small

distance, L. Suppose the best match reading to the right is *n* characters, $x_0x_1...x_{n-1}$. Then we encode the next n + 1 characters $x_0x_1...x_n$ as the triple (L, n, x_n). Associative coding is stationary because the context sorting loses the relative position of the context, as with BWT.

Popular, fast compression programs such as COMPRESS, PKZIP, and GZIP use Ziv-Lempel (LZ) compression (Bell, Witten, and Cleary, 1989). In the LZ77 variant (used by GZIP), strings that occur more than once are replaced with pointers (offset and length) to previous occurrences. This exploits the cache effect because closer strings can be encoded using fewer bits for the offset, corresponding to a higher probability. However, LZ77 wastes code space whenever a string could be coded in more than one way. For example, the third occurrence of *can* in (1) could be coded as a pointer to either the first or the second occurrence, and we need some space to encode this arbitrary choice. LZ78 (used by COMPRESS) eliminates this redundancy by building a dictionary and coding repeat strings as pointers to dictionary entries. (Thus, it is stationary). This does not entirely eliminate redundancy because we could still find more than one encoding by choosing different token boundaries. For instance, we could encode *can* using three pointers to three one-character entries. Although COMPRESS, PKZIP, and GZIP compression is relatively poor, part of the reason is that they sacrifice compression for speed and memory economy. (ACB also codes redundantly, but still achieves good compression). The top compressors use large amounts of memory (16-64 MB or more) and are many times slower.

The rest of this paper is organized as follows. In section 2, we describe the architecture of the PAQ1 archiver, a bit-level predictor using a weighted combination of models and an arithmetic encoder. In sections 3 through 6, we describe four model components: a nonstationary order-8 n-gram model, a long string matcher for n > 8, a word bigram model for English ASCII text, and a fixed length record model. In section 7, we compare PAQ1 with top ranked compression programs. In section 8, we isolate the effects of the PAQ1 components. In section 9, we summarize and suggest future directions.

2. The PAQ1 Architecture

PAQ1 is a predictive arithmetic encoder, like PPM, except that it uses a binary alphabet to simplify encoding and model merging. The idea is to encode an input string *s* as a binary number *x* such that for any randomly chosen string *r*,

$$p(r < s) \le x < p(r \le s) \tag{2}$$

assuming some ordering over the strings. The range of *x* satisfying (2) has magnitude p(s), thus we can always encode a number that satisfies the inequality with at most $\log_2 p(s) + 1$ bits, which is within one bit of the Shannon limit (Shannon and Weaver, 1949).

Howard and Viller (1992) describe efficient methods for encoding *x* in O(|s|) time, where |s| is the length of s in bits. The method used by PAQ1 is to estimate p(s) one bit at a time as the product of the conditional probabilities $p(s_1)p(s_2|s_1)p(s_3|s_1s_2)...p(s_n|s_1s_2s_3...s_{n-1})$. We keep track of the range of x satisfying (2), initially [0,1), and as each bit is predicted, we divide the range into two parts in proportion to p(0) and p(1). Then when the bit is input, we the corresponding segment becomes the new range. When the range becomes sufficiently narrow that the leading base-256 digits become known (because they are the same for the lower and upper bound), then they are output. The point where the range is split is rounded so that only 4 digits (32 bits) of the bounds need to be kept in memory. In practice, this rounding adds less than 0.0002 bits per character to the compressed output.

The advantage of using bit prediction is that it allows models that specialize in different types of data to be integrated by using a weighted combination of their predictions. In PAQ1 there are five models, each outputting a probability and a confidence. This is represented as a pair of non-negative counts, c_0 and c_1, which expresses the idea that the probability that the next bit will be a 1 is $p(1) = c_1/c$ with confidence (weight) c, where $c = c_0 + c_1$. Equivalently, it represents a set of c independent experiments in which 0 occurred c_0 times and 1 occurred c_1 times. The models are combined simply by adding their counts, i.e. $p(1) = \Sigma\, c_1 / \Sigma\, c$ with the summation over the models.

Because it is possible to have p(1) = 0 or p(1) = 1, which would imply an infinitely long code if the opposite bit

actually occurs, we include a bland model to ensure that this does not happen. The bland model is $c_0 = 1$ and $c_1 = 1$, which expresses $p(1) = 1/2$ with confidence 2. The bland model by itself neither compresses nor expands the input.

3. The Nonstationary n-gram Model

PAQ1 guesses the next bit of input by finding a sequence of matching contexts in the input and recording the sequence of bits that followed. An n-gram model, with n = 1 through 8, consists of the previous n - 1 whole bytes of context, plus the 0 to 7 bits of the partially read byte. For example, suppose we wish to estimate p(1) for the fourth bit of the final *t* in *cat* in (1). The 3-gram context is *ca*,011, consisting of the last *n* - 1 = 2 whole characters, plus the bits read so far. This context occurs four times prior to the current occurrence (both *n* and *t* begin with the bits 011). In the first three occurrences (*can*), the next bit is 0. In the fourth (*cat*), the next bit is 1. Thus, the next-bit history in this context is 0001.

The next question is what to do with this history. As mentioned, PPM, BWT, and associative coding models would assume that the outcomes are stationary and independent, and weight all bits equally for p(1) = 1/4. However, from section 1 we saw that the most recent data ought to be given greater weight. Some possibilities are:

- Temporal bigram model. Each event is considered in the context of the previous event. Since no events have yet occurred in the context 1, p(1) = 0.5.
- Inverse reaction model. Only the last occurrence of each event is counted, with weight 1/*t*, where *t* is the age. The age of the last 1 is 1, and the age of the last 0 is 2. Thus, p(1) = 1/(1 + 1/2) = 0.67.
- Inverse temporal model. Each event with age *t* is weighted by 1/*t*, or 1/4, 1/3, 1/2, 1. Thus, p(1) = 1/(1/4 + 1/3 + 1/2 + 1) = 0.48

These models require different amounts of state information to be kept by the source. A stationary, independent source is stateless. A temporal bigram model requires one bit of state information to save the last bit. An inverse reaction model requires saving not only the last event, but also the number of times in a row that it occurred. An inverse temporal model is the most complex. It must save a complete history so that each event weight can be discounted by 1/*t* after time *t*. This model is not stationary because it requires unbounded memory, although it can be approximated by a sum of exponentially decaying counts over a wide range of decay rates.

There is evidence that the brains of animals may use an inverse temporal model or inverse reaction model. Classical conditioning and reinforcement learning are consistent with both models. According to Schwartz and Reisberg (1991), the rate of learning in classical conditioning is inversely proportional to the time between the conditioned stimulus (a signal, such as a dog hearing a bell) and the unconditioned stimulus (the dog gets meat, causing it to learn to salivate when it hears the bell). Also, the rate of learning in reinforcement conditioning is inversely proportional to the time between the behavior being taught (response) and the reward or punishment. In both cases, the significance of the first event (signal or response) is in inverse proportion to its age. These results suggests that this property holds in many types of data that is important to our survival, or else our brains would not have evolved this way.

One problem with all temporal models is that they give incorrect results when the data really is stationary and independent, such as a random source. The more we weight the model in favor of the most recent events, the smaller the sample from which to estimate the probability, and the less accurate the result. For instance, an inverse reaction model can never give a probability between 1/3 and 2/3, even if the true probability is 1/2.

Without knowing the answer, PAQ1 uses an ad-hoc solution. It is temporal in that it favors recent events, although for five or fewer events it is equivalent to a stationary model. It also weights the model in favor of long runs of zeros or ones, which indicate that the estimate p(1) is highly reliable. The rule is:

> If the training bit is *y* (0 or 1) then increment c_y (c_0 or c_1). (3)
> If $c_{1-y} > 2$, then set $c_{1-y} = c_{1-y} / 2 + 1$ (rounding down if odd).

Table 3 shows the state c_0 and c_1 on input 00000001111, and output p(1) when combined with a bland model.

Input	c_0	c_1	p(1)+bland	weight
0000000	7	0	1/9 = 0.11	10
00000001	4	1	2/7 = 0.28	9
000000011	3	2	3/7 = 0.43	7
0000000111	2	3	4/7 = 0.57	7
00000001111	2	4	5/8 = 0.62	8

Table 3. Predictor state and output for the training sequence 00000001111

This rule has the property that when the outcome is highly certain, when p(0) or p(1) is near 1, the weight is increased, roughly in proportion to 1/p(0)p(1). This is because the only way for *c* to become large is to have a long run of zeros or ones.

We use a compromise between a stationary and inverse temporal model because we do not know *a-priori* whether the data is stationary or not. On random data, which is stationary, rule (3) expands the data by about 6.6%. We could reduce the expansion by biasing the model more toward a stationary one, for example:

If the training bit is *y* (0 or 1) then increment c_y (5)
If $c_{1-y} > 4$ then $c_{1-y} = 3c_{1-y}/4 + 1$

But in practice, this hurts compression on most real data.

PPM models normally use a 256 symbol alphabet. They generally use the longest context for which at least one match can be found in the input, up to some maximum (usually about n = 5). However, this was found to give poor compression using a binary alphabet, probably because information from the lower order contexts is never used. Instead, PAQ1 combines all eight n-gram models (for n = 1 to 8) with weight n^2. If we make some inverse temporal assumptions about the data, then n^2 is the predictive value of the context divided by its frequency. The predictive value is n because the event of a mismatch between the current and matched context last occurred n characters ago, so the next character should mismatch with probability 1/n. The frequency of the context is proportional to 1/n because if the probability of a mismatch is 1/n then the expected length of the match is about 2n. If half of all matches of length n match for 2n symbols, then frequency is inversely proportional to length. A number of other weightings were tried (e.g. 1, n, n^3, 2^n, hand tuning for each n, etc.) and n^2 gives the best compression on most files.

To conserve memory, PAQ1 represents the counts c_0 and c_1 using a single 8-bit state. This is possible because both counts cannot be large at the same time, and because large counts can be approximated without significant loss of compression. The counts 0 through 10 are represented exactly. The other possible values are spaced further apart as they get larger: 12, 14, 16, 20, 24, 28, 32, 48, 64, 96, 128, 256, and 512. When a large count needs to be incremented from *a* to *b* where $b - a > 1$, then the increment is implemented by setting the count to *b* with probability $1/(b - a)$. This representation was found to compress only 0.05% worse on typical data than using an exact representation. A pseudo-random number generator (based on the recurrence $r_i = r_{i-24}$ XOR r_{i-55} and a fixed seed) is used to ensure platform independence.

PAQ1 stores the counts for 2^{23} (8M) contexts in a 16 MB hash table. Each table element is two bytes: one byte for the counter state, and one byte for a hash checksum to detect collisions. The context is hashed to a 32-bit value, in which 23 bits are used to index the hash table and 8 bits are used as a checksum. Collisions are resolved by trying 3 successive locations to find a matching checksum or unused element ($c = c_0 + c_1 = 0$), and if none are found, then the element with the lowest total count (n) is replaced. Using an 8-bit checksum means that collision detections are missed with probability 1/256, but this was found to have a negligible effect on compression.

4. The String Matching Model

We could expand the n-gram model to higher n, but this wastes memory because we need to store bit counts for long contexts, most of which may never reoccur. For each byte of input, we need to add 8 hash table entries, or 16 bytes total. Instead we could store the input in a buffer, and search for matching contexts and compute the bit counts on

the fly. Because long matching contexts are highly predictive and likely to agree with each other, it is sufficient to find just one match (the most recent) and increase the weight with the assumption that there may be others. PAQ1 uses a weight of $3n^2$ (values of $2n^2$ to $4n^2$ work well) for a match of n bytes.

PAQ1 uses a rotating 4 MB (2^{22} character) buffer and an index of 1M (2^{20}) 3-byte pointers indexed by a hash of the last 8 bytes with no collision resolution. If a match is found, PAQ1 continues to use it until an unmatched bit is found. Otherwise the current context is compared with the location pointed to by the index to look for a match. The index is updated after each character is read. The total size of the model is 7 MB.

5. The Word Bigram Model

Teahan and Cleary (1997) and Jiang and Jones (1992) note that compression of English text improves when the symbol alphabet is taken to be whole words rather than letters. PAQ1 uses two word n-gram models for n = 1 and 2. A word is defined as a sequence one or more letters (a-z) up to 8 letters long (truncating the front if longer), and case insensitive. For the unigram model, the context is a hash of the most recent whole or partial word, the previous byte (whether or not it is alphabetic, and case sensitive), and the 0 to 7 bits of the current byte read so far. The weight is $(n_0 + 1)^2$ where n_0 is the length of the word. The bigram model also includes the previous word, skipping any characters in between. Its weight is $(n_1 + n_0 + 2)^2$ where n_1 is the length of the previous word. For example, in (1), the unigram context of the final character is *ca* (with 4 matches and weight 9) and the bigram context is *cat, ca* (with no matches and weight 49). The weights are approximately the same as the n-gram model when the spaces between words are included.

The bit counts associated with each context are stored in a hash table of 2^{22} (4M) counter pairs as in the nonstationary n-gram model, for a total size of 8 MB.

6. The Fixed Length Record Model

Many types of data are organized into fixed length records, such as databases, tables, audio, and raster scanned images. We can think of such data as having rows and columns. To predict the next bit, we look at the sequence of bits in the column above it, and optionally at the bits preceding it in the current row as context.

PAQ1 looks for evidence of fixed length records (rows), indicated by an 8 bit pattern (not necessarily on a byte boundary) that repeats r = 4 or more times at equally spaced intervals. If it finds such a pattern with row length $k \geq 9$ bits (not necessarily on a byte boundary), then the following two contexts are modeled: a zero order context consisting only of the column number, and an 8-bit context plus the column number. Their weights are 4 and 16 respectively. The weights were determined in an ad-hoc manner, but in keeping with the general principle that longer or less frequent contexts should receive more weight.

To detect the end of the table, PAQ1 sets a timer to rk (rows times columns) when a table is detected, which expires after rk bits are read with no further detection. It is possible that another table may be detected before the first expires. If the new table has a higher value of rk than the current value, then the new values of r and k take effect.

For example, suppose the input is:

```
$ 3.98\n
$14.75\n
$ 0.49\n
$21.99\n
```

where \n is the linefeed character. The model records for each 8 bit sequence such as "$", ".", or "\n" the position of the last occurrence, the interval *k* to the occurrence before that, and the number of times *r* in a row that the interval was the same. In this example, *r* = 4 and *k* = 56 bits for each of these three patterns. This is sufficient to establish a cycle length of *k* good for the next *rk* = 448 bits (4 lines) of input unless a higher *rk* is seen in the meantime.

The next bit to predict is the first bit of the next line, right under the "$". For the weight-4 model, the context is

simply the bit position in the file mod $k = 56$. In this context, the bit counts c_0 and c_1 come from the first bit of the "$" in the columns above. Since the first bit of "$" is 0, we have $c_0 = 4$, $c_1 = 0$ for $p(1) = 0$ with confidence 16. For the weight-16 model, the context is a hash of "\n" and the bit position mod 56. Again, this occurs 4 times, so $c_0 = 4$, $c_1 = 0$, for $p(1) = 0$ with confidence 64. We need both contexts because (for example), only the first context is useful for predicting the decimal point character.

The zero bit context is stored in a table of counters (as in the nonstationary n-gram and word models) of size 2^{12} (2K). The 8-bit contexts are stored in a hash table of 2^{15} (32K) entries, for a total size of 68K bytes.

7. PAQ1 Performance

PAQ1 was tested against popular and top ranked compressors on the Calgary corpus, a widely used data compression benchmark, and on the Reuters-21578 corpus. Three tests were performed. In the first test, the programs compressed the Calgary corpus files separately into a single "solid" archive (allowing compression across files, if supported), or 14 separate files if the program does not support archiving. The total size of the compressed files is reported. In the second test, the 14 files of the Calgary corpus were concatenated (alphabetically by file name) into a single file of size 3,141,622 bytes. In the third test, the 22 files of the Reuters-21578 collection (Lewis, 1997) were concatenated (in numerical order) into a single file of size 28,329,337 bytes. This corpus consists of 21,578 English news articles in SGML format. The tests were performed on a 750 MHz Duron with 128 KB L2 cache and 256 MB memory under Windows Me. Run times (in seconds) are for compression for the first test only.

PAQ1 was compared with the top 5 programs ranked by ACT, the top 3 ranked by Ratushnyak, and with the winner of the Calgary Corpus Challenge (Broukhis, 2001). For each program, options were selected for maximum compression within 64 MB of memory according to the documentation for that program. In some cases, maximum compression depends on the data being compressed, which requires experimentation with different combinations of options. When the best options differ for different input files, all combinations used are reported.

In the Calgary challenge, the top program is an unnamed submission (which we call DEC) made by Maxim Smirnov on Aug. 14, 2001 for a small cash prize. The challenge requires only the submission of the compressed corpus and a decompression program (that must run within 24 hours on a Windows PC with 64 MB memory), which is evaluated by its combined size after the program and data is compressed by an archiver from a list supplied by Broukhis. The reason that the decompression program itself is included is because the rules allow part or all of the compressed data to be included in the program. The challenge has two parts, one program to extract the entire archive (solid), and one to decompress files read from standard input (split archive). The submitter must demonstrate general purpose compression by compressing an arbitrary file of 0.5 to 1 MB supplied by the challenge to within 10% of GZIP and decompressing it with the split version of the submitted program. Details of DEC are not available, but Smirnov is also the author of PPMN (Smirnov 2002). PPMN details are not available either, but its options indicate it is an order-9 PPM encoder that performs file analysis to automatically select optimizations for text, run length encoding, "e8 transform", and "interleaved coding". The e8 transform is an optimization for DOS/Windows .EXE files in which relative jump/call addresses are translated to absolute addresses.

The test results are shown in Table 1, along with COMPRESS, PKZIP, and GZIP. In the table, the type is LZ (77 or 78), PPMD, PPMZ, BWT (Burrows-Wheeler), AC (associative coding), or NS (nonstationary n-gram). -s indicates a solid archive (compression across files). MB is the memory usage in megabytes where known. Time is seconds to compress the 14 files of the Calgary corpus (decompress for DEC because no compressor is available). Sizes are for the compressed file or files after compressing the Calgary corpus as 14 files or concatenated into a single file, and to compress the concatenated Reuters-21578 corpus. The meanings of the options are as follows:

- GZIP -9: maximum compression
- ACB u: maximum compression.
- BOA -m15 -s: 15 MB memory (maximum), solid archive.
- PPMD/PPMONSTR e -m64 -o8/16/128: encode, 64 MB memory, order 8/16/128. -o1: order 64 context length for var. H. Note: lower orders use less memory, thus give better compression for Reuters.
- SBC c -m3 -b16: compress, maximum compression (adds 12 MB memory), 16*512K BWT block size (8 MB

blocks compressed independently, memory usage is 6 times block size).

- RK -mx3 -M64 -ft: maximum compression, 64 MB memory, filter for text (overriding automatic detection).
- PPMN e -O9 -M:50 -MT1: encode, PPM order 9 (maximum), 50 MB memory (maximum), text mode 1 (best compression)

Program	Options	Type	MB	14 files	Time	Concat.	Reuters
Original size				3,141,622		3,141,622	28,329,337
COMPRESS		LZ78	<1	1,272,772	**1.5**	1,318,269	10,406,527
PKZIP 2.04e		LZ	0.4	1,032,290	**1.5**	1,033,217	8,334,457
GZIP 1.2.4	-9	LZ77	<1	1,017,624	2	1,021,863	8,114,057
ACB	u	AC-s	<16	766,322	110	769,363	4,567,213
BOA 0.58b	-m15 -s	PPMZ-s	15	747,749	44	769,196	4,326,917
PPMD H	e -m64 -o16	PPMII	64	744,057	5	759,674	4,286,536
	e -m64 -o8	PPMII	64	746,316	5	762,843	3,850,243
SBC 0.910	c -m3 -b16	BWT	65	740,161	4.7	819,027	4,197,501
PPMONSTR H	e -m64 -o1	PPMII	64	719,922	13	736,899	4,465,502
	e -m64 -o8	PPMII	64	726,768	11	744,960	3,917,444
RK 1.02 b5	-mx3 -M64	PPMZ-s	64	707,144	44	750,744	4,404,256
	-mx3 -M64 -ft	PPMZ-s	64	717,384	44	750,744	4,207,960
RK 1.04	-mx3 -M64	PPMZ-s	64	712,188	36	755,872	3,978,800
	-mx3 -M64 -ft	PPMZ-s	64	730,100	39	755,876	3,857,780
PPMN 1.00b1	-M:50 -MT1 -O9	PPM	50	716,297	23	748,588	3,934,032
PPMONSTR Ipre	-m64 -o128	PPMII	64	696,647	35	**703,320**	4,514,279
(3/3/02)	-m64 -o8	PPMII	64	704,914	30	707,581	**3,781,287**
DEC-SPLIT		?	<64	685,341	(27 to decompress)		
DEC-SOLID		?-s	<64	**680,558**	(27 to decompress)		
PAQ1		NS-s	48	716,704	68.1	716,240	4,078,101

Table 4. Compression results for PAQ1 and popular or top ranked compression programs on the Calgary corpus (14 files or concatenated) and the Reuters-21578 corpus.

The best compression on the Calgary corpus is 680,558 bytes by the solid version of DEC as a HA (Hirvola, 1993) archive. The archive decompresses to 11 data files with a total size of 672,739 bytes, plus the 24,576 byte decompression program, DEC.EXE. It is possible to compress DEC.EXE to 7056 bytes with PAQ1 (better than HA, RK or PPMONSTR) which further reduces the overall size to 679,795 bytes, or 1.731 bits per character (bpc). The split version is a 685,341 byte HA archive containing 14 compressed files with a total size of 677,624 bytes, plus a 24,576 byte executable, compressible to 6892 bytes by PAQ1 for a total size of 684,516 bytes. No compression program is available, nor are any details of the algorithm used.

Among general purpose programs, the best on the Calgary corpus is PPMONSTR var. Ipre dated 3/3/02, which is currently undergoing development and not yet released as of this writing. Among released compressors, RK 1.02b5 with the -mx3 (maximum compression) option, at 707,144 bytes. Only PPMONSTR Ipre, RK (1.02b and 1.04) and PPMN beat PAQ1, which compresses to 716,704 bytes, or 2.9% larger than the 696,647 bytes for PPMONSTR Ipre. RK and PAQ1 both produce solid archives, using statistics from other files. However, when the files are concatenated, PAQ1 has the best compression of all programs except PPMONSTR Ipre. Concatenation hurts compression in all programs except PAQ1, which compresses to a slightly smaller 716,240 bytes (1.824 bpc). PAQ1 concatenates the input files internally. so the slight difference is due to a smaller header.

On the Reuters corpus, RK, PPMD, and PPMONSTR beat PAQ1. The smallest is 3,781,287 bytes (1.068 bpc) by order-8 PPMONSTR Ipre vs. 4,078,101 bytes for PAQ1 (7.8% larger).

7.1. Calgary Corpus Detailed Results

Table 5 shows compression on individual files in the Calgary corpus for the top compressors, RK 1.02, PAQ1,

PPMONSTR H, and DEC-SPLIT, with options set for maximum compression and 64 MB memory as in Table 4. For RK and PAQ1 sizes are shown for both solid and split archives. For solid archives, sizes shown are as reported by the programs and do not include the archive header. DEC also produces solid archives, but this information was not available. RK analyzes the input files and reorders them prior to producing a solid archive. For consistency, the same ordering was used for PAQ1. The table also tests the effect of reordering in RK order on the concatenated corpus vs. concatenation in alphabetical order by file name as in Table 4.

	Original	Solid archive		As separate files			
File	Size	RK 1.02	PAQ1	RK 1.02	PAQ1	PPMONSTR	DEC-SPLIT
PAPER2	82199	**22111**	22323	22204	22352	**21842**	**21147**
PAPER1	53161	13085	**12783**	14748	14521	**14258**	**14020**
BOOK1	768771	**201950**	206539	**202424**	209195	205165	**195338**
BOOK2	610856	139591	**132151**	138624	137672	**136077**	**132832**
OBJ1	21504	**9397**	10027	9488	9463	**9422**	**9092**
GEO	102400	**47960**	52244	**47740**	51329	53236	**46879**
OBJ2	246814	**63754**	66416	**61932**	64151	65131	**63595**
PIC	513216	**30595**	46975	**30684**	45973	45157	**24217**
BIB	111261	24146	**23011**	24236	24293	**23334**	**23471**
PROGL	71646	13692	**12661**	13260	12708	**12470**	**12688**
PROGC	39611	11614	**10571**	11456	10824	**10721**	**10737**
PROGP	49379	10207	**9048**	9252	**8725**	**8609**	8944
TRANS	93695	14273	**13465**	14696	**14146**	**13747**	14360
NEWS	377109	104435	**97917**	105124	102078	**100771**	**100304**
Total	3141622	**706810**	716386	**705860**	727430	719940	**677624**
Concatenated RK		750500	**716159**				
Concatenated alpha		750740	**716240**				

Table 5. Calgary corpus results on individual files for top compressors. Best results are shown in bold.

The most notable differences between compressors is that RK and DEC outperforms PAQ1 and PPMONSTR on the binary files OBJ2 and GEO, and especially PIC. OBJ2 is a Macintosh executable organized into 16-bit words. GEO contains seismic data organized into 32-bit words. PIC is a black and white image of a page in a textbook (text in French and a line diagram) at 200 dots per inch, organized as a 1728 by 2376 bitmap. RK has a delta filter which replaces each 8, 16, or 32 bit word with the difference from the previous word. On text files, DEC is best, followed by PPMONSTR, PAQ1 and RK.

It is interesting to note that producing a solid archive did not help RK at all! The solid archive is 0.1% larger than the 14 separately compressed files. On the other hand, the solid PAQ1 archive is 1.5% smaller. Nor does the file order make much difference. When the files are concatenated in RK order vs. alphabetical order, the output is only 0.03% smaller for RK and 0.01% smaller for PAQ1. This seems rather insignificant when we consider that the loss of file boundary information expands RK output by 6.2% and PPMONSTR by 2.3%, but does not hurt PAQ1 at all. Using solid archives, PAQ1 compresses better than RK for 8 of 14 files and better than PPMONSTR H for 7 of 14 files.

7.2 Confusion Tests

The next set of experiments tests for specialized models in the top compressors by modifying some of the files in ways that should confuse these models without changing overall entropy. The files PIC, GEO, and BOOK1 from the Calgary corpus were used, since it is possible that the top compression programs might be tuned to this widely used benchmark, and these files are representative of the different data types in this corpus.

The first test was on PIC, a 1728 x 2376 image with one bit per pixel. The image was cropped by removing 8 pixels from the right edge, i.e. removing every 216'th byte. All of the removed bytes are 0. PAQ1, PPMONSTR (-o1, order 64), and PPMN all compress either file to about 45 KB, but this trivial change caused the compression by RK 1.02 to go from 30K to 48K. This suggests that RK has a model specific to PIC. (Malcolm Talyor later confirmed that RK transposes the rows and column to get better compression).

The second test was on GEO, which contains seismic data in 32-bit records. Most of the records consist of large blocks where every record has the form (in binary)

x10000xx xxxxxxxx xxxxxx00 00000000

In other words, the first hex digit is 4 or C, and the last byte is 0. The x's indicate apparently random bits, although there is some correlation between adjacent records in the leftmost bits. An n-gram model would miss this because of the intervening random data. The experiment was to remove this obstacle by de-interleaving the file. Every fourth byte was sampled to produce four smaller files, which were concatenated. The result was that the first 1/4 of the file contains bytes of the form x10000xx, and the last 1/4 contains all zeros. This improves compression by PPMONSTR from 53 KB to 49 KB, and similarly in PPMN, but hurts RK, which goes from 47 KB to 49 KB. The effect in PAQ1 is to render the fixed record model useless, but this is offset by bringing the correlated bytes together as in PPMONSTR, so the compression remains about 51 KB.

The third experiment was to perform a simple substitution cipher on BOOK1 to test for models specific to ASCII text. The cipher is to multiply each byte by 3 (mod 256). For instance, *a* (97) became # (3 x 97 = 35 (mod 256)). This hurt all three compressors. PPMONSTR was affected only slightly, going from 205 KB to 207 KB. RK went from 202 KB to 213 KB, and PAQ1 went from 209 KB to 219 KB. This suggests that the word model in PAQ1 helps compression of English text about as much as the text filter (-ft) in RK. PPMN, which had the best compression, suffered the most, going from 197 KB to 212 KB.

File	Original	PAQ1	RK 1.02	PPMONSTR	PPMN	DEC-split
PIC	513,216	45,973	30,684	45,157	46,583	24,217
PIC cropped	510,840	45,953	48,252	45,200	46,596	
GEO	102,400	51,329	47,740	53,236	52,272	46,879
GEO de-interleaved	102,400	51,259	49,844	49,638	49,726	
BOOK1	768,771	209,167	202,428	205,165	197,181	195,338
BOOK1 ciphered	768,771	219,503	213,548	207,853	212,792	

Table 6. Model confusion test results on top compressors. DEC-split is shown for reference.

It should be noted that if we substitute the cropped PIC into the Calgary corpus results in table 4, then the top five compressors from best to worst becomes DEC, PPMN, PAQ1, PPMONSTR, and RK. Although RK and PPMONSTR outperform PAQ1 on most files, PAQ1 has the advantage that it can make better use of cross file statistics in producing a solid archive.

8. The Compression-Memory Tradeoff

From table 4 we can see that the best compression programs tend to use the most memory. The modular design of PAQ1 allows us to easily combine models and adjust the memory usage by each one to optimize compression for some data set. PAQ1 combines two very general models, the n-gram model and long string matching model, with two specialized models, one for English text and one for fixed length records, common in binary data.

Table 7 shows the effects of removing models or decreasing their available memory on compression of the Calgary corpus. For the n-gram and word models, the memory was adjusted by changing the size of the hash table. The other models were simply added or removed. The word model generally improves the compression of text files, while expanding other files very slightly. The fixed length record model improves compression mostly for GEO, and to a lesser extent PIC and OBJ2, while expanding other files very slightly. Even without the two specialized models,

PAQ1 still achieves very good compression, 731 KB on the concatenated Calgary corpus, compared to 736 KB for the next best compressor, PPMONSTR.

n-gram	string	word	fixed	Size	Time	
-	-	-	-	3,141,622		
1 MB	-	-	-	843,819	46.3	
8 MB	-	-	-	768,463	45.9	
16 MB	-	-	-	758,456	45.6	
32 MB	-	-	-	751,734	45.3	
32 MB	7 MB	-	-	731,637	48.5	
32 MB	7 MB	8 MB	-	723,092	62.1	
16 MB	7 MB	8 MB	68 KB	720,310	71.2	
32 MB	7 MB	4 MB	68 KB	717,766	68.0	
32 MB	7 MB	8 MB	68 KB	716,704	68.1	(PAQ1)

Table 7. The effects of removing model components or reducing memory on compression of the concatenated Calgary corpus.

9. Conclusion

PAQ1 demonstrates that nonstationary bit-level modeling is comparable in speed, memory, and compression to the best PPM and BWT implementations. All of these algorithms exploit a very general (but not universal) redundancy found in many types of data -- that some n-grams are more common than others. What they don't exploit -- and PAQ1 does -- is that these common n-grams tend to cluster together. This comes at a cost, because optimizing any compression algorithm for one type of data hurts compression for other types. PAQ1 gets very good compression on mixed data, but at the cost of expanding homogeneous data, including random data. It is as if the model keeps looking for patterns where none exist.

PAQ1 uses a nonstationary n-gram model, which is a compromise between a stationary model where all events are equally significant, and an inverse temporal model where the significance of an event is inversely proportional to the time since it occurred. The latter is motivated by studies of learning in animals. The models are weighted by n^2, which we argued is equal to the predictive power of an n-gram divided by its frequency if we assume that the n-grams themselves are generated by an inverse temporal process. Finally, the models are weighted by 1/p(0)p(1) because probabilities near 0 or 1 are more useful than probabilities near 1/2.

The modular design of PAQ1 makes it easy to add specialized models for certain types of data, such as ASCII text or fixed length records. We found experimental evidence that RK contains similar models, as well as one for images of exactly the same width as PIC from the Calgary corpus. We could add more models to improve compression on standard benchmarks even further, but there is little point in doing so. The DEC program illustrates the extent to which this approach can be taken.

I believe that advances in compression will come in two areas. First is the addition of more memory. Second, it will come from a better understanding of specialized types of data. For data that is meaningful to humans, such as text or movies, the problem is closely related to artificial intelligence. In lossy multimedia compression, the problem is to discard what we cannot see or hear, and keep what we can, which implies a deep understanding of the process of human perception. Lossless text compression is a problem in natural language processing. To see this, imagine the problem of compressing the following hypothetical dialog from the Turing test for AI (Turing, 1950):

```
Q: Please write me a sonnet on the subject of the Forth Bridge.
A: Count me out on this one. I never could write poetry.
Q: Add 34957 to 70764.
A: (Pause about 30 seconds and then give as answer) 105621.
Q: Do you play chess?
```

```
A: Yes.
Q: I have K at my K1, and no other pieces.  You have only K at K6 and R at R1.
It is your move.  What do you play?
A: (After a pause of 15 seconds) R-R8 mate.
```

Since arithmetic encoding is known to be optimal, compression quality depends entirely on knowing the probability p(*s*) for any given input string *s* whatever the source, for example, human communication. This implies that for any given question *q* and answer *a*, we could calculate p(*a*|*q*) = p(*qa*)/p(*q*). Knowing p(s), we could therefore generate answers as above with distribution p(*a*|*q*). Since humans also produce answers with distribution p(*a*|*q*) (by our original assumption about the source), the program's answers would be indistinguishable from that of a human, and the program would pass the Turing test.

Of course we have a long way to go before data compression programs can learn from a corpus of text to play chess or simulate human errors in arithmetic, but there are some things we can do. For instance, Rosenfeld (1996) has demonstrated improved compression (actually word perplexity, a function of entropy) in models that learn semantic associations, such as *sonnet* and *poetry*, based on their close proximity in running text, so that the occurrence of either word predicts the other. (Bellegarda et. al., 1996) has gone further by exploiting the transitive property of semantics (using a process called latent semantic analysis, essentially a 3-layer linear neural network) to learn such associations even when the words do not appear near each other, but both appear near other words such as *rhyme*.

PAQ1's word bigram model uses a fixed lexical model to find words, which it defines as a sequence of letters from *a* to *z*. This would not work in foreign languages that use extended character sets, or in languages that lack spaces between words, such as Chinese. The problem also occurs in Finnish verbs, German nouns, and with English suffixes such as -s, -*ing*, -*ed*, etc. However, Hutchens and Alder (1997) demonstrated that lexical knowledge is learnable from an n-gram model: the conditional entropy of characters outside the boundary of a word is higher than for characters within the boundary. Essentially, this is because words have a higher frequency than non-words, and because words cannot overlap.

PAQ1 source code was released on Jan. 7, 2002 under the GNU general public license, and is available at `http://cs.fit.edu/~mmahoney/compression/`. Future versions will use incompatible archive formats, thus they will have different names: PAQ2, PAQ3, etc. I anticipate that future versions will incorporate some of the language modeling ideas discussed here. One of my goals is to bridge the "lexical boundary" between the word-level semantic and syntactic models used by the statistical NLP community (primarily speech recognition) and the n-gram character models of the data compression community.

Acknowledgments

I wish to thank Dmitry Shkarin for helpful comments on PPMD/PPMONSTR, pointing out Maxim Smirnov's PPMN program, and providing a pre-release of PPMONSTR var. I. I also thank Malcolm Talyor for comments explaining RK, and Szymon Grabowski for comments explaining PPMN's "e8" option and PKZIP's memory requirements, and fixing a link to BOA.

References

Arnold, Ross, and Tim Bell (1997), "A corpus for the evaluation of lossless compression algorithms", Snowbird UT: Proc. Data Compression Conference, http://corpus.canterbury.ac.nz/resources/report.pdf

Bell, Timothy, Ian H. Witten, John G. Cleary (1989), "Modeling for Text Compression", ACM Computing Surveys (21)4, pp. 557-591, Dec.

Bellegarda, Jerome R., John W. Butzberger, Yen-Lu Chow, Noah B. Coccaro, Devang Naik (1996), "A novel word clustering algorithm based on latent semantic analysis", Proc. IEEE Intl. Conf. on Acoustics, Speech, and Signal Processing, vol. 1, 172-175.

Bloom, Charles (1998), "Solving the Problems of Context Modeling",

http://www.cco.caltech.edu/~bloom/papers/ppmz.zip

Broukhis, Leonid A. (2001), The Calgary Corpus Compression Challenge, http://mailcom.com/challenge/

Burrows, M. and D. J. Wheeler (1994), “A Block-sorting Lossless Data Compression Algorithm“, Digital Systems Research Center, http://gatekeeper.dec.com/pub/DEC/SRC/research-reports/abstracts/src-rr-124.html

Buyanovsky, George (1997), ACB v2.00c., http://www.alberts.com/authorpages/00013300/prod_50.htm, http://www.cbloom.com/news/bygeorge.html

Clarkson, P. R., and A. J. Robinson (1997), “Language model adaptation using mixtures and an exponentially decaying cache”, IEEE ICASSP, Vol. 2, 799-802.

compress 4.3d for MSDOS (1990), ftp://ctan.tug.org/tex-archive/tools/compress/msdos.zip

Gilchrist, Jeff (2001), Archive Comparison Test, http://compression.ca/

Gailly, Jean-loup, GZIP 1.2.4 (1993), http://www.kiarchive.ru/pub/msdos/compress/gzip124.exe

Hirvola, H. (1993), HA 0.98, http://www.webwaves.com/arcers/msdos/ha098.zip

Howard, Paul G., and Jeffrey Scott Viller (1992), "Practical Implementations of Arithmetic Encoding", in *Image and Text Compression*, ed. James A. Storer, Norwell MA: Klewer Academic Press, pp. 85-112. http://www.cs.duke.edu/~jsv/Papers/HoV92.actech.pdf

Hutchens, Jason L., and Michael D. Alder (1997), "Language Acquisition and Data Compression", 1997 Australasian Natural Language Processing Summer Workshop Notes (Feb.), pp. 39-49, http://ciips.ee.uwa.edu/au/~hutch/research/papers

Jiang, J., S. Jones (1992), “Word-based dynamic algorithms for data compression”, IEE Proc. Communication, Speech, and Vision, 139(6): 582-586.

Lewis, David (1997), The Reuters-21578 Text Categorization Test Collection, distribution 1.0, http://www.research.att.com/~lewis/reuters21578.html

Mäkinen, Sami J. (2001), SBC v0.910, http://www.geocities.com/sbcarchiver/

PKWARE Inc. (1993), PKZIP version 2.04e, ., http://www.pkware.com/

Ratushnyak, A. (2001), The Art of Lossless Data Compression, Vol. 22t, http://www.geocities.com/SiliconValley/Bay/1995/texts22.html

Rosenfeld, Ronald (1996), “A Maximum Entropy Approach to Adaptive Statistical Language Modeling”, Computer, Speech and Language, 10, http://www.cs.cmu.edu/afs/cs/user/roni/WWW/me-csl-revised.ps

Schindler, Michael (1997), “A Fast Block-sorting Algorithm for Lossless Data Compression”, 1997 Data Compression Conference, http://www.compressconsult.com/szip/

Schwartz, Barry, and Daniel Reisberg (1991), *Learning and Memory*, New York: W. W. Norton and Company.

Shannon, Claude, and Warren Weaver (1949), *The Mathematical Theory of Communication*, Urbana: University of Illinois Press.

Shkarin, Dmitry (2002), PPM: One step to practicality, Proc. 2002 Data Compression Conference, http://www.dogma.net/DataCompression/Miscellaneous/PPMII_DCC02.pdf

PPMD and PPMONSTR var. H are available at ftp://ftp.elf.stuba.sk/pub/pc/pack/ppmdh.rar

Smirnov, Maxim (2002), PPMN version 1.00 beta 1 release N1, http://msmirn.newmail.ru/

Sutton, Ian (1998), boa 0.58 beta, http://sac-ftp.externet.hu/pack4.html

Taylor, Malcolm (1999, 2000), RK Software, http://rksoft.virtualave.net/index.html

Teahan, W. J., John G. Cleary (1997), “Models of English text”, IEEE Proc. Data Compression Conference, 12-21.

Turing, A. M., (1950) "Computing Machinery and Intelligence“, Mind, 59:433-460, http://www.loebner.net/Prizef/TuringArticle.html