

# PPM Model Cleaning

Milenko Drinić<sup>†</sup>, Darko Kirovski<sup>‡</sup>, and Miodrag Potkonjak<sup>†</sup>

<sup>†</sup> Computer Science Department, University of California, Los Angeles, CA

<sup>‡</sup> Microsoft Research, One Microsoft Way, Redmond, WA

## Abstract

The Prediction by Partial Matching (PPM) algorithm uses a cumulative frequency count of input symbols in different contexts to estimate their probability distribution. Excellent compression ratios yielded by the PPM algorithm have not instigated broader use of this scheme mainly because of its high demand for computational resources. In this paper, we present an algorithm which improves the memory usage by the PPM model. The algorithm heuristically identifies and removes portions of the PPM model which are not contributing toward better modeling of the input data. As a result, our algorithm improves the average compression ratio up to 7% under the memory limitation constraint at the expense of increased computation. Under the constraint of maintaining the same level of compression ratios, our algorithm reduces the memory usage up to 70%.

## 1 Introduction

Prediction by Partial Matching (PPM) has set the standard in lossless data compression with respect to the compression ratio since Moffat's 1990 implementation of PPMC [6]. The original algorithm that was presented in 1984 by Cleary and Witten [3] evolved over the years through numerous improvements. PPM algorithms achieve superior performance by using finite-context statistical modeling. The stream is encoded using an arithmetic coder. For each of the contexts they maintain a separate adaptive conditional probability distribution of input symbols. A PPM algorithm collects input symbol frequencies as the input is processed, which results in the adaptiveness of probability distributions. A collection of probability distributions for all contexts of the same length constitutes a probability model for that context length. A PPM algorithm combines these models into a single model that is used for coding. The blending of different probability models into a single one is achieved through the insertion of artificial *escape* symbols at the transition from a probability model corresponding to a longer context to a shorter one. *Escape* symbols are inserted when a particular model does not contain the input symbol. Despite producing excellent compression ratios, PPM based compressors were far less common in practice because of their high demand for memory and computational resources.

### 1.1 Problem description

One of the main characteristics of the PPM algorithm is that it is adaptive. It collects statistics for many different contexts which makes the algorithms memory intensive. The memory requirement of a straightforward PPM algorithm implementation is  $O(M^{K+1})$  in the worst case, where  $M$  is the cardinality of the alphabet of input symbols, and  $K$  is the maximum context length. In terms of the length of an input data

sequence  $n$ , PPM algorithms in the worst case require  $O(n^2)$  memory. Upon receiving and processing every symbol from the input stream, the PPM model is updated. If the upcoming input symbol exists in the current PPM model, its frequency is updated. In the case when the upcoming symbol is new to the current context, a new entry is created in all applicable contexts. The update of the PPM model is non-selective. Regardless of the frequency of the further usage of the created entries, they affect the probability estimates of the rest of the upcoming symbols. Because of the practical limitations of Arithmetic and Logic Units (ALUs), symbol frequencies are rescaled when the largest frequency reaches a certain limit. This improves the locality of the PPM model on a current portion of the input data stream. Rescaling also reduces the effect of the seldom-occurring symbols on the overall probability estimate. However, this effect cannot be neglected even after rescaling because the probability prediction of *escape* symbols depends also on the number of distinct symbols in a particular context. We identify entries that are not contributing toward better modeling of the input data as *pollution*, and portions of the PPM model that are not used for the encoding as *polluted regions*. Polluted regions of considerable size increase the total amount of used memory.

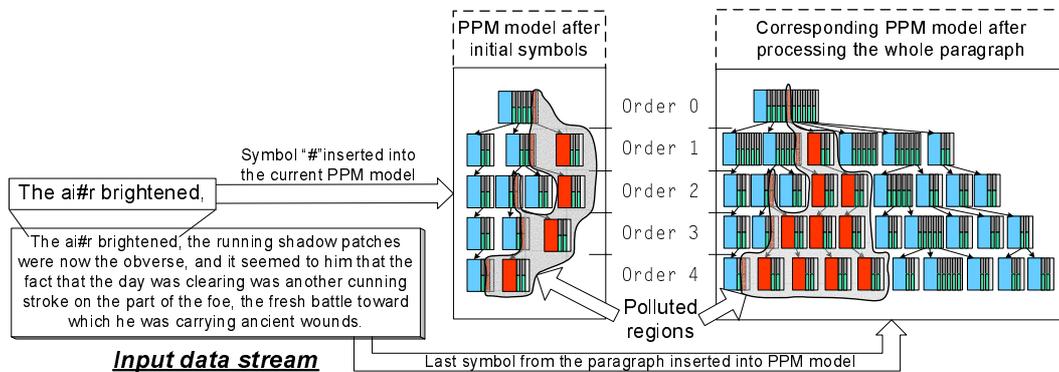


Figure 1: An example of the PPM model pollution.

Consider the case when a particular symbol rarely appears in the input data stream. In the extreme case: the symbol appears only once, close to the beginning of the input data stream. For example, a PPM compressor, with a maximum context length of 4, processes a text file with a typographical error where the symbol “#” is inserted by mistake. After the symbol is processed, it is inserted into the PPM model by creating new entries in the appropriate contexts. This is illustrated in Figure 1 with new contexts which create the polluted region after several initial symbols. The PPM model is also expanded with new entries with the symbol “#” in the already existing contexts. These entries reduce the probability estimate of reoccurring symbols in the text, and negatively affect the overall compression ratio. Therefore, they are included in the polluted region. Moreover, the PPM model tree is expanded with new contexts which include the special symbol and the corresponding combinations of input symbols in its vicinity of the maximum context length. These new contexts, even though never used in the remainder of the processing of the input data stream, require a share of memory which increases the total memory usage.

The center of attention of most improvements of the PPM model are directed toward direct improvement of escape probability. We are focusing our attention on improving the PPM model after it is built by a certain convention for frequency updating of regular and *escape* symbols. We describe a novel approach for improving the generic PPM paradigm from the perspective of its memory consumption at the expense of increased computation. We have developed an algorithm that removes the polluted contexts and the polluted entries in other contexts. We show that by cleaning the PPM model when

the memory limit is reached, we are able to improve significantly compression ratios. In cases when the available memory is limited, obtained compression ratios with cleaning are in the range of compression ratios when the files are compressed without cleaning but with a double memory limit. The cleaning of PPM models exposed numerous possibilities for further improvements for building the probability model. Our cleaning algorithm also introduced the need for new structures for the representation of PPM models such that the navigation throughout the different contexts and context lengths is faster.

## 2 Related Work

Arithmetic coding is based on the unpublished Elias algorithm which is described by Abramson [1]. After Rissanen [8] generalized and resolved accuracy issues of the algorithm, arithmetic coding became practically feasible. A standard strategy for improving the speed of an arithmetic coder is to replace computationally expensive multiplications and divisions with shifting and additions [10], or pre-computed look up tables [5]. Improvements for the storage of cumulative counts is described in [7].

The largest impact on the performance of a PPM algorithm is the probability estimate of the *escape* symbol. Most of the different variants of PPM differ by their predetermined policy<sup>1</sup> adopted for *escape* symbol probability estimates. The original PPM algorithm was presented by [3]. Moffat's careful implementation of the C variant of the PPM algorithm [6] has made PPM the compression engine to beat. Other improvements of fixed maximum context length PPM algorithms include variants D [4] and X [11]. An alternative to those approaches is the construction of an additional model for *escape* estimation that is dependent on a set of parameters associated with the current context [9]. An unbounded maximum context length algorithm referred to as PPM\* was introduced by [2]. A comprehensive summary of PPM and all of its modifications and improvements can be found in [12].

## 3 PPM Overview

Arithmetic coding is used in many lossless data compression schemes because of the excellent compression provided by a sophisticated probability model. Arithmetic coding has low complexity, and it is relatively easily implemented. However, it requires computationally expensive multiplications and a division for each of the input symbols. Let's denote the input data stream as  $x^n = (x_1, x_2, \dots, x_n)$ ,  $x_i \in \mathcal{A}$ , where  $x^n$  is a sequence of  $n$  symbols from the finite and discrete source alphabet  $\mathcal{A}$ . An arithmetic coder sequentially processes each symbol  $x_i$ ,  $i \in \{1, 2, \dots, n\}$ , of the input data stream based on the conditional probability  $p(x_i|x^{i-1})$ . The encoder and decoder operate independently from one another. In order to successfully encode and decode all of the symbols, both the encoder and decoder have to maintain the same probability models, and use the same conditional probability  $p(x_i|s)$  for each symbol  $x_i$  given a sequence  $s$ .

Prediction by partial matching (PPM) uses finite-context models of symbols [3] to estimate the probability distribution for upcoming symbols. A *context* of order  $k$  is a sequence of  $k$  consecutive symbols of the input data stream. The context of order  $k$  of the current symbol  $x_i$  is the sequence of symbols  $c_i^k = (x_{i-k}, x_{i-(k-1)}, \dots, x_{i-1})$ . Similarly, a context of any  $k$  symbols is denoted as  $c^k$ . The set of all possible contexts of length  $k$  can be represented as an  $M$ -ary tree with depth  $k$ , where  $M = |\mathcal{A}|$ . Given

---

<sup>1</sup>In this case, a predetermined policy implies a policy that is independent of the current state of PPM model.

the maximum context length  $K$ , the PPM model is an  $M$ -ary context tree of depth  $K$  where each context  $c^k$ ,  $k \leq K$ , contains a set of entries of symbols  $\alpha \in \mathcal{A}$  that were previously seen in  $c^k$ . Each of the entries contains a count of the number of appearances of the corresponding symbol in  $c^k$ . The conditional probability  $p(\alpha|c^k)$  denotes the probability estimate of symbol  $\alpha$  being found in the context  $c^k$ , and it is determined based on the frequency counts of all symbols seen in context  $c^k$ . For cases when the symbol  $\alpha$  is not previously seen in  $c^k$  (i.e.  $p(\alpha|c^k) = 0$ ), the PPM model applies the *escape* mechanism for switching between longer to shorter contexts. At the root of the PPM context tree is the special order -1 context which contains all symbols from the alphabet  $\mathcal{A}$ . This way, the problem of estimating the probability prediction of novel events is translated to context switching from longer contexts to shorter ones until the input symbol is found. An example of the PPM model with maximum context length of 2 after processing the input string "shareware" is shown in Figure 2.

Finally, the input symbol  $x_i$  found in context  $c_i^k$  when the starting context is of length  $L \leq K$  is encoded with the following probability:

$$p(x_i|c_i^L) = \left[ \prod_{j=k+1}^L p(esc|c_i^j) \right] \cdot p(x_i|c_i^k). \quad (1)$$

Note that the input symbol  $x_i$  is encoded with a probability estimate of  $p(x_i|c_i^L)$  because the cumulative counts of all previously encountered symbols,  $x_i$ , is *de facto* encoded with  $p(x_i|x^i)$ .

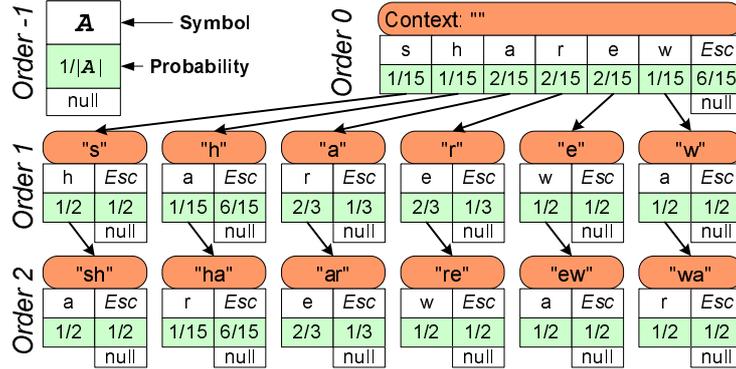


Figure 2: A detailed example of the PPM model with a maximum context length of 2 after processing the word "shareware". For each context, corresponding entries and their probability estimates are shown. The *escape* probability estimates are calculated using variant A modeling [3].

After processing each symbol, the frequency counts for the already existing entries and *escape* symbol are updated. Also, new entries and contexts are added if needed. A standard improvement of the PPM model is through an exclusion mechanism. The set of symbols in the entries for context  $c^k$  is a subset of a set of symbols of  $c^{k-1}$  where  $c^k = c^{k-1}\alpha$ . When an input symbol  $x_i$  is being encoded and is not found in  $c_i^k$ , and the context is switched to  $c_i^{k-1}$ , both the encoder and decoder can exclude from the calculation of conditional probability  $p(x_i|c_i^{k-1})$  all symbols encountered in  $c_i^k$ . Such exclusion always improves the overall compression ratio because more probability is assigned to symbols exclusively seen in  $c_i^{k-1}$  and not in  $c_i^k$ . Since a higher probability estimate implies shorter code, the overall compression ratio is improved.

### 3.1 PPM implementation details

The implementation of the PPM algorithm used throughout our experiments is a bounded maximum context length heuristic that is developed based on the following observations and criteria.

**Escape symbol probability estimate.** Our implementation takes into account three different parameters of the current context for calculation of the *escape* symbol probability. A context that contains a large number of distinct symbols should have a low probability *escape* estimate. When the number of distinct symbols in a context approaches the cardinality of the alphabet  $\mathcal{A}$ , the probability estimate of *escape* symbols converges to 0. Another important parameter is the relative distribution of counts inside a context. The value quantifying the frequency distribution count is calculated by comparing the largest count and the total count in the context. If the largest count is close to the total count, the context is dominated by a single symbol, i.e. it is more likely that the symbol with the maximum count will be correctly predicted than to emit the *escape* symbol. The third parameter used for calculation of the *escape* probability estimate is the total count of symbols in the context. Simply, if the context has a high total count, it appeared frequently, which implies that there is a smaller likelihood of emitting the *escape* symbol in such a context.

**Deterministic context escape probability estimate.** The deterministic context is a context that contains only a single symbol. It has been shown ([2]) that deterministic contexts require special attention for the calculation of the *escape* symbol probability estimate. Deterministic context almost always has a lower likelihood of emitting the *escape* symbol than any of the variants of the PPM algorithm would predict. The difference is not linear, and it is highly dependant on the type of data being processed. We have adopted an adaptive pre-computed table for calculation. The table has a fixed scale which enables the arithmetic coder to use shifting instead of a computationally expensive division operation, while not losing accuracy because the scale is a power of 2.

**Memory usage.** The PPM algorithm is known for its large memory consumption. In the worst case, the PPM model could grow to  $O(M^{K+1})$ , where  $M = |\mathcal{A}|$ , and  $K$  is the maximum context length. We have limited the total memory used by the compression algorithm. When the limit is reached, the whole model is discarded, and the model is reset to the initial state.

**Rescaling the frequency counts.** The limited precision arithmetic of ALUs mandate a periodical scaling down of the frequency counts. When a symbol with the maximum frequency reaches a certain limit, all entries in the context scaled down by a factor of 2. Aside of preventing an overflow during coding, this procedure helps to improve locality of the encoded symbol. In our implementation, the limit when scaling occurs is determined empirically rather than on a fixed parameter depending on the ALU precision.

**Exclusions.** A standard improvement to the PPM algorithm is the exclusion of symbols seen in a higher context order from the calculation of the probability estimate. *Update exclusions* is a technique which updates only the frequency count for the context in which the symbol is found rather than all shorter contexts. This technique improves the compression ratio of bounded maximum context length variants of PPM by approximately 2% [12]. It also improves the processing speed since updating the symbol includes only the update of already passed entries.

## 4 Cleaning of the PPM Model

The PPM model requires large amounts of memory for processing the input data stream. Many practical implementations of the PPM algorithm, including ours, set the memory limit to a certain value. When the model exceeds the limit the whole model is discarded, and the model is reset to the initial state. The purpose of cleaning the PPM model is to identify and remove parts of the PPM model that are not contributing to better probability prediction, i.e. to remove polluted regions from the PPM model. Cleaning of the PPM model reduces the memory used by the model. The side-effect of reduced memory usage is the reduction of the frequency of resetting of the PPM model. Consequently, improving the compression ratio of the input data stream.

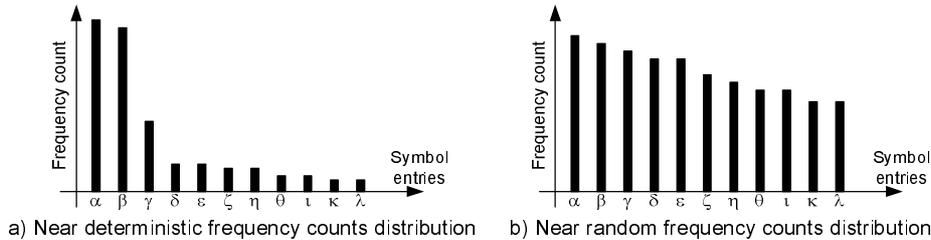


Figure 3: An example of two different distributions of frequency counts in a context.

### 4.1 Definition of pollution

Consider two examples of distributions of frequency counts in a context in the moment when we decide to clean the model shown in Figure 3. Case a) depicts a situation where the distribution is near deterministic. Frequency counts for two entries clearly dominate in the context. The remaining entries have low frequency counts and they most likely do not contribute to better modeling of input stream. The removal of those entries would potentially improve the overall compression ratio. Another extreme case is shown in Figure 3 b) where frequency counts are almost evenly distributed across all entries. No entry can be singled out as the one that dominates the remaining entries. Moreover, the count of each individual entry is sufficiently high such that it can be assumed that symbols corresponding to the entries will appear in the future in the same context.

We heuristically define pollution entries of the context as those entries that are not contributing to the better modeling of the input data stream. The entry for symbol  $\alpha_j$  with frequency count  $f_{\alpha_j}$  from context  $c^k$  is considered as a polluted entry if all of the following conditions are satisfied:

$$\begin{aligned}
 (i) \quad & f_{\alpha_j} < \gamma \cdot f_{\alpha_D} \\
 (ii) \quad & f_{\alpha_j} < \frac{\gamma}{2} \cdot f_{max} & f_{\alpha_i} \leq f_{\alpha_m}, \quad \forall i < m \\
 (iii) \quad & \sum_{i=1}^j f_{\alpha_i} < \delta \cdot \sum_{i=1}^D f_{\alpha_i}
 \end{aligned} \tag{2}$$

where  $D$  is cardinality of context  $c^k$ ,  $f_{\alpha_D}$  is the largest frequency count in context  $c^k$ , and  $f_{max}$  is the limit of the frequency count when rescaling occurs. Parameters  $\gamma$  and  $\delta$ ,  $0 < \gamma, \delta < 1$ , are determined empirically. The above heuristic conditions can also be expressed as follows: (i)  $f_{\alpha_j}$  is dominated by the maximum frequency count in the context  $f_{\alpha_D}$ ; (ii)  $f_{\alpha_j}$  is sufficiently smaller than the maximum allowed frequency  $f_{max}$ ; and (iii) the cumulative count of all frequency counts that are smaller or equal to  $f_{\alpha_j}$  in  $c^k$  are sufficiently smaller than the total frequency count for  $c^k$ . When all the above conditions are met, the entry for  $\alpha_j$  is considered polluted and it is selected for removal. Note that with these criteria, at least one entry  $\alpha_D$  will not be selected for removal.

We also heuristically define pollution contexts as those contexts that are seldom or never used during the compression of the given data stream. A context  $c^k$  is a candidate to be considered polluted if one of the following conditions is true: (i)  $c^k = c^{k-1}\alpha_j$  where  $\alpha_j$  is an entry in the context  $c^{k-1}$  that is considered polluted; (ii)  $c^k = c^m\alpha_j c^{k-m-1}$  where  $\alpha_j$  is the polluted entry from  $c^m$ ; and (iii)  $c^k = c^l c^{k-l}$  ( $c^{k-l}$  is suffix of  $c^k$ , formally denoted as  $c^{k-l} \sqsubset c^k$ ) and  $c^{k-l}$  is considered polluted. A context that is a candidate to be considered polluted is declared polluted if the following condition is satisfied:

$$\sum_{i=1}^D f_{\alpha_i} < (K - (m - 1)) \cdot \frac{\gamma}{2} \cdot f_{max}, \quad \alpha_i \in c^k \quad \forall i. \quad (3)$$

The condition for removal of a whole context  $c^k$  is the low total count of all the frequency counts in  $c^k$ . The condition becomes more restrictive as the order of contexts increases since the cleaning algorithm should not remove high order contexts that can have lower count and still contribute to better probabilistic modeling. Because of the update exclusions procedure (see Section 3.1), a situation may arise where a longer context contains entries whose frequency counts is larger than those in shorter contexts for the same symbol.

## 4.2 The cleaning algorithm

The cleaning algorithm is designed in such a way that a valid PPM model is maintained after the algorithm is applied. The first step of our algorithm is the identification of polluted entries. The tree containing the PPM model is traversed starting from the maximum context length. Each context  $c^k$  is considered separately and a number of entries identified as pollution. Once entry  $\alpha_j$  is identified as polluted, all of its successors contexts  $c^m$  ( $c^m = c^k\alpha_j c^{k-m-1}$ ) need to be verified if they are polluted or not. In addition, all contexts  $c^l$ , such that  $c^m \sqsubset c^l$  and  $c^m$  is a verified polluted context, are also verified as polluted. The context  $c^k$  is removed from the PPM model only if it is verified as polluted, that does not have any entry whose successors are not verified as polluted, and that it is not a suffix to any of the contexts that is not verified as polluted. When all successor contexts of the entry for  $\alpha_j$  are verified as polluted, the entry can be removed from the context  $c^k$ . Note that a deterministic context cannot be removed by the entry removal criteria. The pseudo code for the procedure described above is shown in Figure 4.

```

Initialize  $k = K$ 
Repeat
  For all  $c^k$ 
    Select entries  $\alpha_j$  for removal by conditions from Equation (2)
    For all selected entries  $\alpha_j$ 
      For all  $c^m | c^m = c^k\alpha_j c^{k-m-1}$  and  $c^l | c^m \sqsubset c^l$ 
        Verify pollution of all  $c^m$  and  $c^l$  by the criterion from Equation (3)
        Remove all verified  $c^m$  and  $c^l$ 
      If all  $c^m$  and  $c^l$  removed
        Remove entry  $\alpha_j$ 
    Update  $k = k - 1$ 
Until  $k = 0$ 

```

Figure 4: Pseudo-code for the procedure of cleaning the PPM model

We have identified the following decision strategies when the cleaning of the PPM model will be applied: (i) when regular rescaling occurs, (ii) periodically, and (iii) instead

of resetting the model to the initial state. Performing the cleaning procedure every time when a context needs to be rescaled results in high frequency of cleaning. Such a strategy is inefficient because the model does not become polluted enough for cleaning to be effective. Moreover, the entries that are not polluted cannot be distinguished from the pollution so the cleaning procedure removes them as well. Alternatively, the cleaning algorithm can be modified such that it is applied only to the portion of the PPM tree connected to the context being rescaled. However, that strategy would fail to clean large polluted portions of the PPM model which are not used in encoding. The second strategy applies the cleaning procedure periodically with an upfront determined frequency of cleaning. The third strategy applies cleaning whenever the amount of memory used for the PPM model exceeds a certain limit. Many practical implementations, including ours, when the assigned memory is exceeded, reset the model to the initial state. Instead of resetting the model back to the initial state, we have applied cleaning of the model. We have performed experiments with all strategies described above. The third strategy produced the best results which are presented in Section 5. The first strategy has not produced results that are better, on average, than the strategy without cleaning. The second strategy produced a modest improvement in terms of compression ratios when a small amount of memory is available.

## 5 Experimental Results

We tested the effectiveness of our approach on the standard set of Calgary corpus extended with a dynamic link library file, an executable file, and two pdf files. Additional files represent common types of data whose compression typically yields poor compression ratios. They are larger than files from Calgary corpus, and require larger amounts of memory for compression. Our PPM compressor yields the average compression ratio on the Calgary corpus of 2.36 bits per character (bpc), which is in the approximate range of PPM\*C compressor [2]. On the full set of benchmarks, our algorithm yields 3.11 bpc. All experiments were performed using a maximum context length of 4. The detailed compression ratios are shown in Table 1. The third column presents bpc for PPM\*C compressor. The fourth column in Table 1 represents the bpc for compression without cleaning and without a memory limit. Because of the chosen cleaning strategy, the fourth column also represents the compression ratios with since the memory limit is never reached in this case. The next six columns show the bpc for three cases of memory limit (ML): 256KB, 512KB and 1MB. For each of the memory limitation cases, we compressed the files from the benchmark suite without cleaning, and with cleaning when there is no memory available for the increase of the PPM model. The average bpc over the whole set of benchmarks used is shown in the last two rows in Table 1.

Figure 5 presents the average compression ratios of the extended benchmark suite with respect to the assigned memory. Our algorithm yields an improvement of 5.9% or 0.21 bpc in terms of the average compression ratio over the extended corpus when the memory limit is set to 128KB. At the same time, the improvement over Calgary corpus is 7.7% or 0.22 bpc. For each of the cases of a particular memory limit, the compression ratios when the PPM model is cleaned are better than the compression ratios with double the memory limit without cleaning. In other words, under the limitation of maintaining the same level of compression ratio, our algorithm is able to reduce the memory usage up to 70%. The average compression ratio obtained with cleaning of the PPM model converges toward the average compression ratio without cleaning as the memory limit increases. When the memory limit increases, the PPM model is cleaned less often as the limit is reached less frequently. Finally, at the point when none of the files reaches the memory limit (16MB), the cleaning of the model does not occur, and

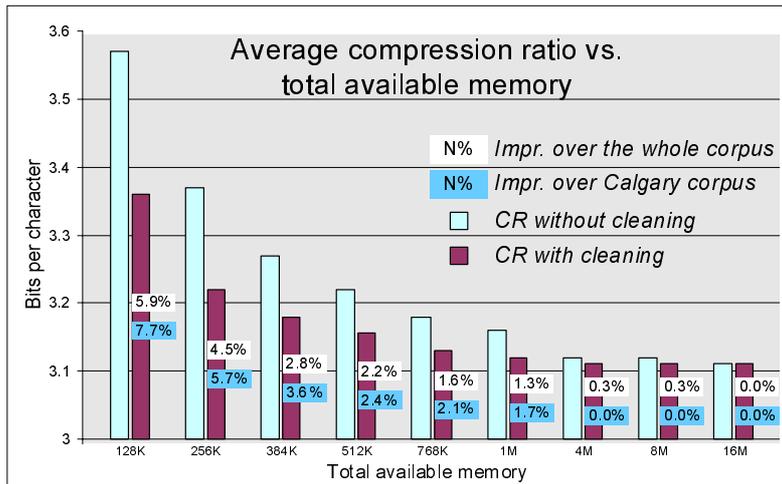


Figure 5: Average compression ratios of the extended benchmark suite plotted against the total available memory.

the compression ratios for the two cases are identical. The improvement of compression ratios is a consequence of the longer usage of a quality PPM model while compressing the input stream. Model is cleaned when the memory limit is reached, which extends the lifetime of the PPM model. Also, the prediction probabilities that contribute to the better modeling of the input stream are preserved while pollution is removed.

Our implementation of the PPM algorithm compresses the whole benchmark suite in 15.8 seconds (the Calgary corpus in 2.4 seconds) using a maximum context length of 4 without cleaning the PPM model on a 933MHz Pentium III computer. Since we used data structures inherited from standard implementations of the PPM algorithm, a search through the PPM context tree requires a significant amount of time. This problem is exacerbated when the PPM model is large. In such cases the amount of time needed to encode the whole benchmark suite is prohibitively long. In the case when the memory is restricted to 320KB, runtime of the algorithm over the whole benchmark suite is 105 seconds. Alternative data structures which would enable simpler navigation through contexts can reduce the run-time overhead of the cleaning algorithm considerably.

## 6 Conclusion

We presented an algorithm that improves the usage of memory by the PPM algorithm by improving the compression ratios under constrained memory resources. Our cleaning algorithm identifies and removes portions of the PPM model that pollute the probability distributions, and have a negative impact to the overall compression ratio. Simultaneously, probability predictions that contribute toward a better PPM model are preserved, and the lifetime of the PPM model is extended. As a consequence, when there is a hard memory limit, our algorithm yields compression ratios of the range as if the available memory is more than doubled. The main drawback of our algorithm is that it slows down the compression procedure significantly. The data structure used in our implementation is inherited from the previous PPM algorithm implementations and it does not support operations from our algorithm in a runtime efficient manner.

File name	File size [bytes]	Compression ratio [bits per character]								
		PPM*C	ML:Unlimited		ML:256KB		ML:512KB		ML:1MB	
			w/o & w/	w/o	w/	w/o	w/	w/o	w/	w/o
bib	117,541	1.91	1.95	2.33	2.18	1.95	1.96	1.95	1.95	
book1	785,393	2.40	2.29	3.05	2.66	2.74	2.46	2.46	2.33	
book2	626,490	2.02	2.02	2.65	2.36	2.34	2.16	2.12	2.04	
geo	102,400	4.83	4.50	4.73	4.60	4.63	4.54	4.56	4.52	
news	387,168	2.42	2.48	3.20	2.89	2.94	2.70	2.71	2.53	
obj1	21,504	4.00	3.78	3.78	3.79	3.78	3.78	3.78	3.78	
obj2	246,814	2.43	2.55	2.74	2.70	2.67	2.60	2.62	2.60	
paper1	54,411	2.37	2.48	2.69	2.50	2.48	2.48	2.48	2.48	
paper2	83,930	2.36	2.39	2.71	2.44	2.39	2.40	2.39	2.39	
pic	513,216	0.85	0.80	0.81	0.79	0.80	0.79	0.80	0.80	
progc	41,098	2.40	2.48	2.61	2.48	2.48	2.48	2.48	2.48	
progl	73,890	1.67	1.81	1.88	1.80	1.81	1.81	1.81	1.81	
progp	51,345	1.62	1.81	1.81	1.81	1.81	1.81	1.81	1.81	
trans	94,487	1.45	1.73	2.03	1.78	1.73	1.73	1.73	1.73	
dll	550,912	N/A	4.65	4.73	4.63	4.65	4.60	4.61	4.57	
exe	1,163,264	N/A	3.63	3.89	3.83	3.81	3.75	3.74	3.68	
pdf1	1,323,005	N/A	7.35	7.61	7.49	7.56	7.46	7.52	7.46	
pdf2	124,928	N/A	7.35	7.50	7.26	7.42	7.23	7.36	7.26	
Avg (Calgary)		2.34	2.36	2.64	2.49	2.47	2.41	2.41	2.37	
Avg (whole suite)		N/A	3.11	3.37	3.22	3.22	3.15	3.16	3.12	

Table 1: Compression ratios for the extended benchmark suite are shown for three different cases: without memory limit (ML), when ML is 256KB, when ML is 512KB, and when ML is 1MB. For each of the cases, compression ratios are shown for runs of our PPM algorithm without cleaning (w/o), and with cleaning (w/).

## References

- [1] N. Abramson. *Information Theory and Coding*. McGraw Hill, New York, NY, 1963.
- [2] J. Cleary and W. Teahan. Unbounded length contexts for PPM. *The Computer Journal*, 40(2/3):67–75, 1997.
- [3] J. Cleary and I. Witten. Data compression using adaptive coding and partial string matching. *IEEE Transactions on Communications*, 32(4):396–402, 1984.
- [4] P. Howard. The design and analysis of efficient lossless data compression systems. Technical Report CS-93-28, Brown University, 1993.
- [5] P. Howard and J. Vitter. Arithmetic coding for data compression. *Proceedings of IEEE*, 82(6):857–65, June 1994.
- [6] A. Moffat. Implementing the PPM data compression scheme. *IEEE Transactions on Communications*, 38(11):1917–21, 1990.
- [7] A. Moffat, R. Neal, and I. Witten. Arithmetic coding revisited. *ACM Transactions on Information Systems*, 16:256–94, 1998.
- [8] J. Rissanen. Modeling by shortest data description. *Automatica*, 14:465–71, 1978.
- [9] D. Shkarin. PPM: one step to practicality. *Data Compression Conference*, pages 202–11, 2002.
- [10] L. Stuiver and A. Moffat. Piecewise integer mapping for arithmetic coding. *Data Compression Conference*, pages 3–12, 1998.
- [11] I. Witten and T. Bell. The zero frequency problem: Estimating the probabilities of novel events in adaptive text compression. *IEEE Transactions on Information Theory*, 37(4):1085–94, July 1991.
- [12] I. Witten, A. Moffat, and T. Bell. *Managing gigabytes: compressing and indexing documents and images*. Morgan Kaufmann, San Francisco, Ca., 1999.