PPM: one step to practicality

Dmitry Shkarin
Institute for Dynamics of Geospheres, Moscow, Russia
E-mail: dmitry.shkarin@mtu-net.ru

PPM is one of the most promising lossless data compression algorithms using Markov source model of order D. Its main essence is the coding of a new (in the given context) symbol in one of inner nodes of the context tree; a sequence of the special escape symbols is used to describe this node. In the reality, the majority of symbols is encoded in inner nodes and the Markov model becomes rather conventional. In spite of the fact that PPM algorithm achieves best results in comparison with others, it is used rarely in practical applications due to its high computational complexity. This paper is devoted to the PPM algorithm implementation that has complexity comparable with widespread practical compression schemes based on LZ77, LZ78 and BWT algorithms. This scheme has been proposed in [1] and named *PPM with Information Inheritance* (PPMII).

1 Basic definitions

Let A be a discrete alphabet consisting of $M \geq 2$ symbols; $x^n = x_1, \ldots, x_n,\ x_i \in A$ be first n symbols of message; $|\sigma|$ be the length of sequence σ or the cardinality of set σ. The probability of symbol $x_{n+1} = a \in A$ for a source with memory depends on *current context* $s_d = x_n, \ldots, x_{n-d+1} \in A^d,\ d \leq D$. The set of all possible contexts can be presented as nodes in M-ary tree with depth D. Context (sequence) s describes a path from tree root to the current node also denoted as s.

Usually the true conditional probabilities are unknown and the *coding conditional probabilities* $q(a|s)$ depend on characteristics of one or several subsequences $x^n(s)$, $x^n(s)$ is the subsequence of all symbols x_j such that $x_{j-1}, \ldots, x_{j-d} = s_d$. These characteristics are: frequency $f(a|s) = f(a|x^n(s))$ of symbol a in $x^n(s)$, alphabet $A(s) = A(s, x^n(s)) = \{a :\ f(a|s) > 0\}$, its cardinality $m(s) = |A(s)|$ etc. Even at small D, the number of model states is big, subsequences $x^n(s)$ are short in average, and their statistics is insufficient for effective compression.

PPM algorithm [2] is based on an (implicit) assumption: the longer is the common initial part of contexts, the more (in average) similarity there is between their conditional probability distributions. High PPM efficiency means this assumption is fair for the majority of real sources.

Let Z_n be the set of nodes $s_d,\ d \leq D$ on the current branch $x_n,\ x_{n-1},\ \ldots$ of context tree; $s(a)$ be the context with maximum length, for which $f(a|s) > 0$; $d_n(a) = |s(a)|$ (if such context is absent then $d_n(a) = -1$); $q(a|s)$ and $q(esc|s)$ be the conditional probabilities for symbol $a \in A(s)$ and for escape symbol. The escape symbol signals a new symbol appearance in $A(s)$.

The key feature of all PPM modifications is the representation of coding conditional probability for any symbol $a \in A$ as

$$q^*(a|x^n(s)) = \left[\prod_{i=d+1}^{D} q(esc|s_i)\right] \cdot q(a|s_d), \qquad d = d_n(a) \tag{1}$$

The product of escape conditional probabilities describes for the decoder a sequential descent from s_D to $s(a)$ (if $d_n(a) = D$ then this product is equal to 1). Usually

$$q(a|s) = \frac{t(a|s)}{T(s)},\ \forall a \in A^-(s); \qquad q(esc|s) = \frac{t(esc|s)}{T(s)}, \tag{2}$$

where $A^-(s_j) = A(s_j) \backslash A(s_{j+1})$, s_{j+1} is a *child context* of s_j on the current tree branch Z_n (and vice versa, s_j is a *parent context* for s_{j+1}), $t(\cdot|s)$ are *generalized frequencies* introduced in [1], $T(s)$ is the sum of all $t(\cdot|s)$. As a rule, $t(\cdot|s)$ are written as $t(a|s) = f(a|s) + c$, $t(esc|s) = \gamma f(esc|s)$. For example, PPMC [3] and PPMD [4] correspond to $c = 0$, $\gamma = 1$ and $c = -1/2$, $\gamma = 1/2$ respectively.

After encoding of current symbol a, it is necessary to update (to increase by one) $t(a|s)$ in various tree nodes $s \in Z_n$. Early PPM variants [2] used *full updates* when frequencies are changed for all $s \in Z_n$. Later PPM variants [3] use *update exclusions* when frequencies are changed only in nodes with length $|s| \geq d_n(a)$. In some sense, full updates and update exclusions correspond to the two ultimate cases.

During a descent along Z_n with the help of escape symbols, one must eliminate symbols already checked in higher order contexts (*exclusions*). Name such symbols as *masked symbols* and contexts (not) containing such symbols — $(n)m$*-contexts*. Designate s as a *binary context* if $m(s) = 1$ and only two probabilities are used, i.e. $q(a|s) \equiv 1 - q(esc|s)$.

2 Evaluation of generalized frequencies of symbols

1. Information inheritance. The main difficulty of all explicit context-based modeling schemes is the statistics insufficiency in the higher order contexts. Many ways have been proposed to overcome this problem: predictions weighting for lower and higher order contexts (CTW, interpolated Markov model), symbol coding only in contexts that have enough (by some criterion) statistics (LOE, state selection). All these methods require a lot of computational resources and are unacceptable for our purposes. We can take advantage of the similarity of distribution functions in parent and child contexts and set the initial value $t_0(a|s)$ of the generalized symbol frequency in the child context with regard to information about this symbol gathered in the parent context. Such an approach has two virtues: firstly, reference to the parent statistics occurs only at addition of a new symbol to the child context, i.e. rarely enough, that causes existence of linear (not depending on tree depth) time complexity solutions; secondly, due to the rare use of the parent statistics again, the model can quickly adapt to the variations of character of input data.

The following notation is used below: s_i is the new context ($T(s_i) = 0$) or the context, to that new symbol a has to be added ($t(a|s_i) = 0$), s_k is the longest context which contains the current symbol a ($s_k = s(a)$). The addition of a new symbol to the old context statistics will be our initial concern. Locally, at the given point of the encoded text, it would be optimal immediately to use PPM-model of order k, i.e. to reduce the context tree depth to k; in that case, we eliminate errors associated with inaccuracy of the escape probabilities estimation. On the other hand, we need only statistics similar to statistics in s_i, for this reason, we must perform reduction of tree depth only along current context branch Z_n. Such tree depth reduction removes distortions of the probabilities distribution brought in by the symbol masking under update exclusions and increases precision of symbol probability estimation in s_i.

Now, we can equate symbol a probability estimations in s_i and in s_k for tree with pruned branch:

$$\frac{t_0(a|s_i)}{T(s_i) + t_0(a|s_i)} = \frac{t(a|s_k)}{T(s_k) + T_{i,k}}, \tag{3}$$

where $T_{i,k}$ is the statistics gathered in contexts s_j, $k < j \leq i$:

$$T_{i,k} = \sum_{j=k+1}^{i} \left(T(s_j) - t(esc|s_j) - \sum_{n=1}^{m(s_j)} t_0(a_n|s_j) \right)$$

Equation (3) is too complex for calculations and requires an additional variable containing the sum of $t_0(a_n|s_j)$ in each context structure. As a rule, the escape happens to the parent, i.e. $k = i - 1$; if it is not true then statistics gathered in intermediate contexts s_j is small and consists mainly of the initial values $t_0(a|s_j)$. Therefore, we can neglect all intermediate s_j in $T_{i,k}$. We can also replace $t(esc|s_j)$ and $t_0(a_n|s_j)$ by their evaluations in PPMD method. Resolving equation (3) relative to the *inherited frequency* $t_0(a|s_i)$ and using aforesaid simplifications, we get formula:

$$t_0(a|s_i) = \frac{T(s_i) \cdot t(a|s_k)}{T(s_k) - t(a|s_k) + T(s_i) - m(s_i)} \tag{4}$$

At deriving equations (3) – (4), it was implicitly assumed that $t_0(a|s_i)$ estimation is performed after statistics update in s_k. The same argumentation can be repeated for the case when inheritance is performed before statistics update; some intermediate cases are also possible. It requires an introduction of free parameter τ: $\tau = 0$ corresponds to information inheriting *after* statistics update and $\tau = 1$ corresponds to inheriting *before* statistics update. Repeating all previous reasonings as well for the new context ($T(s_i) = 0$), we come to the final formula:

$$t_0(a|s_i) = \tau + \begin{cases} \dfrac{t_0(esc|s_i) \cdot (t(a|s_k) - \tau)}{T(s_k) - t(a|s_k)} & , \ T(s_i) = 0 \\ \dfrac{T(s_i) \cdot (t(a|s_k) - \tau)}{T(s_k) - t(a|s_k) + T(s_i) - m(s_i)} & , \ T(s_i) \neq 0 \end{cases} , \ 0 \leq \tau \leq 1 \, , \tag{5}$$

where $t_0(esc|s_i)$ should be specified by extraneous methods (PPMD, for example). By experiments, the optimal value of τ is approximately 1/4 for PPMD method.

At deriving equation (5), it was supposed that $t_0(a|s_i)$ is calculated immediately after the first advent of a in s_i. However, at large D, it is more preferable to delay calculation of (5) until s_i is encountered next time. In this case, the symbols probabilities distribution in another parent $s'_k = s'(a)$ is usually more similar to the distribution in s_i and calculated $t_0(a|s_i)$ corresponds better to unknown conditional probability of a in s_i. On the other hand, such "delayed" symbol addition to s_i requires an additional searching of a in s'_k and additional searching of s'_k itself; it would lead to the increase of algorithm complexity. For this reason, delayed symbol addition is performed for the new contexts only ($T(s_i) = 0$).

2. Update exclusions modification. For original PPM (without information inheritance), the parent contexts are used only for coding of new symbols which were not seen earlier in the child contexts. Their importance raises when the information inheritance is added, now they are also used for $t_0(a|s)$ calculation. Therefore, the speed of statistics gathering in the parent contexts becomes more important. The conventional update exclusions mechanism does not meet these requirements, the full updates work badly also.

The update exclusions can be modified in the following way: along with $t(a|s_k)$ increment in $s_k = s(a)$, we shall increase the frequency in its parent s_{k-1}, but with weight 1/2. At conventional update exclusions, the symbol frequency and its probability estimation in the parent s_j are proportional to the number of child contexts s_{j+1} containing this symbol. Not to lose completely this property, the statistics update in s_{k-1} must be stopped when $t(a|s_k)$ reached some threshold. Experimentally, it was found equal to 8.

Proposed update exclusions modification requires one additional scanning of s_{k-1} and this scanning can increase the execution time nearly twice in some cases. We can suppose if the symbol was processed by the longest possible context ($d_n(a) = D$)

then statistics in this tree branch is already stable, the escape probability is small and here is no need to update statistics in the parent. With this assumption, the execution time is increased by 2-10% only (it depends on D).

3 Evaluation of escapes probabilities

For estimation of escape probability, we shall divide all contexts into three types: binary contexts, nm-contexts and m-contexts. Consider each type of contexts separately.

1. Escape probability for binary contexts. At big enough D, the symbol encoding begins at binary context in 60 – 80% of cases. The coding probability of a single symbol for such contexts is strongly connected to the escape probability: $q(a|s) = 1 - q(esc|s)$. For these reasons, accurate estimation of $q(esc|s)$ gives significant gain for compression efficiency. We shall construct an additional model for $q(esc|s)$ estimation dependent on some set of parameters $\mathbf{w}(s) = (w_1, \ldots, w_h)$ associated with the current context. Not to confuse this model with the basic model, we shall denote it as *SEE* (*secondary escape estimation*) model [5] and parameters of this model — SEE-contexts. Escape probability for each SEE-context is computed by the usual formula for a mean

$$q(esc|\mathbf{w}) = \langle\delta\rangle = \frac{S(\mathbf{w})}{N(\mathbf{w})}, \quad S(\mathbf{w}) = \sum_{j=1}^{N(\mathbf{w})} \delta_j(\mathbf{w}), \tag{6}$$

where $\delta_j = 1$ when a new symbol has appeared on j-th step and $\delta_j = 0$ otherwise, $N(\mathbf{w})$ is the number of tests.

Scheme with continuous rescaling of statistics is used in PPMII to improve algorithm adaptivity. The number of tests is fixed formally: $N = N_0$ and $(1 - 1/N_0)$ is the scaling factor. On j-th step, δ_j is added to the sum S and the mean value $\langle\delta\rangle$ is subtracted from S, i.e. change of S is $\Delta S = \delta_j - S/N_0$ for each step. N_0 can be chosen as power of two to eliminate division operation, thus, the total number of operations per one estimating-updating cycle is one addition, one subtraction and one arithmetic shift. This simple method of the mean estimation can be applied to various statistical and modeling programs and, actually, it is intensively used in [6].

The number h of $\mathbf{w}(s)$ components, their quantization and their influence on $q(esc|\mathbf{w})$ can be found experimentally only. They are enumerated below in the decreasing influence order.

1) Of course, the escape probability depends on generalized symbol frequency $t(a|s)$ to a great degree. This variable is quantized to 128 values.

2) PPM uses similarity of parent and child contexts, therefore, the alphabet size $m(s_{i-1})$ of the parent has strong effect on the probability of escape from the child s_i. This $\mathbf{w}(s)$ component is quantized to 4 values.

3) The experiments show that highly predictable data blocks are interleaved with not so predictable ones for real sources. The size of such block is small (3–5 symbols), it corresponds to a natural language text segmentation into words and parts of words. The probability of the previous encoded symbol in the previous context is included into $\mathbf{w}(s)$ to trace switching between blocks. This variable is quantized to 2 values.

4) The current coded symbol mostly correlates with the previous symbol. It is inexpedient to include the whole previous symbol into $\mathbf{w}(s)$ because the number of SEE-contexts would be too big and the frequency of each SEE-context would be too small. Only single-bit flag is included into $\mathbf{w}(s)$. The flag value is set to 0 if two higher bits of the previous symbol are zeroed and to 1 otherwise.

5) The *long block* of symbols with length L is the sequence of input symbols for which the escapes to lower orders did not occur and the coding probability for L

symbols of this sequence was larger than 1/2. It is quite probable the PPM model with $D < L$ works badly for such blocks. Special flag is included into $\mathbf{w}(s)$ to signal the occurrence of a long block.

6) By analogy with 4), the flag built of two higher bits of single observed symbol of binary context is added to SEE-context.

Thus, the SEE-model for binary contexts consists of $128 \times 4 \times 2^4 = 8192$ SEE-contexts. Quantization details of various $\mathbf{w}(s)$ components can be found in [7]. The SEE-model performs averaging over large contexts groups, so it depends weakly on statistical outliers at small $T(s)$. Therefore, the best value of the free parameter τ in (5) is $\tau = 1$ and initial estimation of $t_0(esc|s)$ is performed by PPMC: $t_0(esc|s) = 1$.

2. Escape frequency for nm-contexts. This type of contexts is a more difficult case because such contexts occur rarely at high D and an adaptive method (similar to aforedescribed one) would not gather enough statistics. For this reason, the semi-adaptive method is chosen that has some parallel with PPMD method.

Suppose the symbols probabilities distribution in the context is geometrical, i.e.

$$q(a_n|s) = \rho^n(1-\rho), \quad 0 < \rho < 1, \quad n = 0,\ 1,\ 2, \ldots \tag{7}$$

Then, at binary to nonbinary context transformation, the base number ρ can be found with the help of Sec. 3.1 results. At known ρ, the escape frequency $t_{02}(esc|s)$ can be calculated for the context containing two symbols. These calculations can be performed numerically only because symbols may occur not neccessarily in the decreasing probability order.

Furthermore, $t(esc|s)$ value is changed only at addition of a new symbol to the context similarly to PPMD method. The increment of $t(esc|s)$ is

$$\delta_1 = \begin{cases} 1/2, & 4m(s) < m(s(a)) \\ 1/4, & 2m(s) < m(s(a)) \\ 0, & \text{for all other cases} \end{cases} \tag{8}$$

The additional adjustment is required, when the new symbol has a small probability:

$$\delta_2 = 1 - t_0(a|s), \qquad \forall a:\ t_0(a|s) < 1 \tag{9}$$

The final formula is those:

$$t(esc|s) = t_{02}(esc|s) + \sum (\delta_1(s, s(a)) + \delta_2(a, s)), \tag{10}$$

where $t_{02}(esc|s)$ is calculated only once, at binary to nonbinary context transformation, the summation is performed at each new symbol occurrence, δ_1 and δ_2 are defined by equations (8) – (9).

3. Escape frequency for m-contexts. The escape probability for these contexts depends mostly on $T(s)$. This quantity must be presented with very high precision resulting in the large number of SEE-contexts and the low frequency of their occurrence. Therefore, we shall model the behavior of $t(esc|s)$ which depends weakly on the summary frequency, but not $q(esc|s)$. The mean estimation of $t(esc|s)$ is performed similarly to the mean estimation in Sec. 3.1 except for the beginning of coding (at small $N(\mathbf{w})$). At the beginning of coding, the formally fixed number of tests N_0 varies under the law $N_0 = 2^{1+\lceil \log N'(\mathbf{w}) \rceil}$, $N'(\mathbf{w}) = \max(4, N(\mathbf{w}))$, that leads to the faster adaptivity of the mean estimation.

Components of vector $\mathbf{w}(s)$ are enumerated below for this type of contexts:

1) The escape frequency highly depends on $|A^-(s)|$, it is quantized to 25 values.

2) The one bit flag is included into SEE-context that contains the comparison result for $m(s_i) - m(s_{i+1})$ and $m(s_{i+1})$.

3) Similar flag, built of comparison result for $m(s_{i-1})-m(s_i)$ and $m(s_i)-m(s_{i+1})$, is included too.

4) It is the same as 4) item in Sec. 3.1 enumeration.

5) Some correlation does exist between $t(esc|s)$ and the average generalized frequency $\overline{t(s)} = T(s)/(m(s)+1)$; it is quantized to 2 values.

Thus, the total number of SEE-contexts is $25 \times 2^4 = 400$.

4 Implementation details

Let's list the most time expensive operations:

1) searching for the current coded symbol a in the list of all symbols seen in s and estimating the probability interval for this symbol, denote this operation as $prob(s,a)$;

2) escaping to the parent context in the case of symbol a not found in s, denote this operation as $suffix(s)$;

3) finding the next active context after symbol a encoding, denote this operation as $successor(s,a)$, it can be written formally:

$$successor(s,a) = \begin{cases} sa, & |s| < D \\ suffix(s)a, & |s| = D \end{cases} \tag{11}$$

The operation $prob(s,a)$ is the most time consuming one, it can take up to a half of the execution time. The searching of the current symbol is done by the usual linear scan of the array of `TRANSITION` structures. `TRANSITION` structure contains symbol a and its generalized frequency $t(a|s)$. The array is sorted in the decreasing frequency order for the speed-up of the search. More complex search methods would not give any speed gain because, firstly, the alphabet size $m(s)$ is usually small and, secondly, it is necessary to perform exclusions while searching.

The operation $suffix(s)$ can be eliminated by saving a reference to the parent s_{i-1} in `CONTEXT` structure containing characteristics of the context s_i. The operation $successor(s,a)$ is also eliminated by saving the corresponding reference in `TRANSITION` structure. For memory saving, `CONTEXT` structure is created only for repeatedly seen contexts ($T(s) > 0$); the position in the coded string is only stored for new contexts ($T(s) = 0$).

Graphical representation of the used data structures is drawed on Fig.1. The figure shows that the proposed algorithm wastes 12 bytes for each repeated (binary or nonbinary) context and 6 bytes for each transition structure at the nonbinary context. For comparison, one of the most memory efficient PPM/PPM* implementations [8] requires 8 4-byte machine words for the nonbinary context structure, 6 words for the binary one and 6 words for the transition structure at the nonbinary context.

The precision of the frequency representation is 1 for binary contexts and 1/4 for nonbinary ones. The statistics in nonbinary context is scaled (all frequencies are halved) when the value of one of frequencies exceeds threshold 30. Simplified variant of the range coder [9] is used as an entropy coder. The division operation in equation (5) is approximated by series of comparisons, other used approximations can be found in [7].

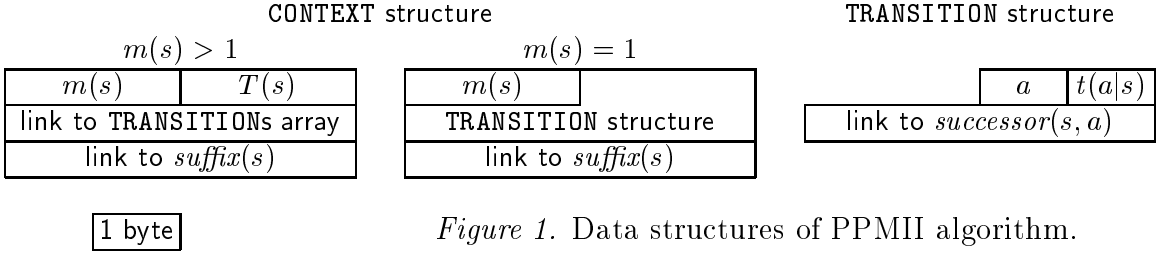

Figure 1. Data structures of PPMII algorithm.

5 Complicated PPMII (cPPMII)

PPMII algorithm demonstrates nice results, so it becomes to be interesting to look at maximal compression efficiency that similar approach can provide. We will not limit ourselves by the requirement of low computational complexity in this section.

Some improvements can be obtained by mere removing of introduced simplifications. Delayed addition of a new symbol to the context (Sec. 2.1) is performed for any contexts, but not just for binary ones. For the update exclusions modification (Sec. 2.2), the statistics is updated at any context length including $|s(a)| = D$ case. Moreover, the statistics is updated for three contexts in the parent-child chain with increments 1/2, 1/4, and 1/8. The precision of the frequency representation is increased up to 1/8. Other improvements require individual considerations.

1. Improving probability estimation for more probable symbols and for less probable ones. Statistics in the parents is accumulated faster than in the "young" child contexts (with small $T(s)$), so there is good reason to use the parent statistics repeatedly.

The generalized frequency of each of more probable symbols (the symbol a_m falls into this group when $q(a_m|s) \geq q(a_{MPS}|s)/2$, a_{MPS} means the most probable symbol (MPS) in the context) is corrected when the child s_i is young ($T(s_i) < T(s_{i-1})$) and symbol a_m probability estimation in s_i is less than the same estimation in s_{i-1}. New value $t_1(a_m|s_i)$ is calculated as a weighted average of $t(a_m|s_i)$ and of the adjusted frequency value $t'(a_m|s_{i-1})$ in the parent:

$$t_1(a_m|s_i) = \frac{T(s_i) \cdot t(a_m|s_i) + T(s_{i-1}) \cdot t'(a_m|s_{i-1})}{T(s_i) + T(s_{i-1})}, \tag{12}$$

where

$$t'(a|s_{i-1}) = t(a|s_{i-1}) \frac{T(s_i) - t(a|s_i)}{T(s_{i-1}) - t(a|s_{i-1})},$$

under conditions ($q(a_m|s_i) \geq q(a_{MPS}|s_i)/2 \cap T(s_i) < T(s_{i-1}) \cap q(a_m|s_i) < q(a_m|s_{i-1})$). The same $t(a_m|s_i)$ correction is performed while inheriting to new contexts.

The probability overestimation of less probable symbol (the symbol a_l falls into this group when $t(a_l|s) < 2$) can have ill effect on $q(a_m|s)$ estimation in the young contexts (the context s falls into this group when $\overline{t(s)} < 4$). Therefore, $t(a_l|s)$ is simply incremented by 3/4, but not by 1:

$$\Delta t(a_l|s) = 3/4, \quad (t(a_l|s) < 2 \cap \overline{t(s)} < 4) \tag{13}$$

2. Improving adaptive mean estimation. The mean estimation method (Sec. 3.1) requires minimal computational resources, but it adapts too slowly at small $N(\mathbf{w})$, therefore cPPMII uses its modification mentioned in Sec. 3.3 everywhere.

Adaptation speed of the mean estimation can be additionally increased if some suppositions are made on the mean dependency of vector $\mathbf{w}(s)$ components. Suppose some variable x (the probability, for example) monotonically depends on the discrete variable i, i.e. $\langle x(i) \rangle \approx (\langle x(i-1) \rangle + \langle x(i+1) \rangle)/2$. Then, at small $N(i)$, we can update statistics not only for i value, but also for $(i-1)$ and $(i+1)$, with some small weight that decreases while $N(i)$ increases. For calculations simplicity, this weight is chosen as equal to $2^{-\lfloor (\log N(i))/2 \rfloor}$ and it is set to zero after exceeding some threshold. This technique is applied to any variable i that is quantized to more than two values.

3. Adaptive probability estimation for MPS. Compression efficiency is markedly affected by the precision of MPS probability estimation, therefore, after frequency correction (see Sec. 5.1), an adaptive probability estimation for MPS is

performed. The adaptive model for MPS is built similar to the model for escape symbols. The behavior of $t(a_{MPS}|s)$ is modeled for nm-contexts and the behavior of $q(a_{MPS}|s)$ is modeled for m-contexts. By analogy with SEE, this method can be called *SSE* (*Secondary Symbol probability Estimation*).

For nm-contexts, vector $\mathbf{w}(s)$ consists of:

1) the generalized symbol frequency $t(a_{MPS}|s)$, it is quantized to 68 values;

2) the one bit flag indicating whether statistics rescaling has been performed;

3) the comparison result of current context length $|s|$ with the average used context length $\langle d_n(a) \rangle$ (averaging is performed over last 128 symbols);

4) the same as 4) and 6) items in Sec. 3.1 enumeration;

For m-contexts, vector $\mathbf{w}(s)$ consists of:

1) the probability estimation $q(a_{MPS}|s)$, it is quantized to 40 values;

2) the comparison result of average masked and nonmasked symbols frequencies;

3) the one bit flag indicating whether only one symbol is not masked;

4) the same as 2) and 4) items in previous enumeration;

4. Additional SEE-contexts components. Additional SEE-context fields for binary contexts are as follows:

1) the comparison result of $m(s_{i-1})$ with $m(s_p)$, where s_p is the previous context;

2) the number of binary parent contexts, it is quantized to 2 values;

3) the flag built of two higher bits of symbol preceding the already encoded symbol;

4) the same as 3) item in the first Sec. 5.3 enumeration;

Two additional $\mathbf{w}(s)$ components are added to the SEE-model for m-contexts. They are the same as 2) items in both Sec. 5.3 enumerations.

5. Adaptive escape frequency estimation for nm-contexts. The SEE-model for these contexts is built similarly to the one for m-contexts, i.e. $t(esc|s)$ is modeled, but not $q(esc|s)$. $\mathbf{w}(s)$ components are as follows:

1) the alphabet size $m(s)$, it is quantized to 25 values;

2) the result of calculation $(4m(s_i) > 3m(s_{i-1}))$;

3) the same as 4) – 6) items in Sec. 3.1 enumeration, 2) – 3) items in the first Sec. 5.3 enumeration and 1) item in Sec. 5.4 enumeration;

6 Experimental results

PPMII algorithm and its complicated modification were implemented on C++ programming language and these implementations are publicly available at [7]. The executable file `PPMd.exe` corresponds to the basic algorithm and `PPMonstr.exe` corresponds to cPPMII. All experiments were carried out on the standard Calgary corpus.

1. Evaluation of contribution of each algorithm part. Unweighted average bits per byte (bpb) for all 14 corpus files are presented in Table 1 for each PPM algorithm modification as described in sections 2–3. Description of each column:

PPMD – initial PPMD by P.G.Howard (implementation of author);
\+ II – previous scheme plus information inheritance (Sec. 2.1);
\+ UEM – previous scheme plus update exclusions modification (Sec. 2.2);
\+ SEE1 – previous scheme plus SEE-model for binary contexts (Sec. 3.1);
\+ EE1 – previous scheme plus escape estimation for nm-contexts (Sec. 3.2);
\+ SEE2 – previous scheme plus SEE-model for m-contexts (Sec. 3.3);

2. Time and memory requirements. In the second experiment, time and memory requirements of PPMII/cPPMII implementations were compared with the widespread practical implementations of LZ77 (ZIP [10]) and BWT (BZIP2 [11]) algorithms and, also, with the most powerful implementation (PPMZ2 [12]) of PPM*

Table 1. PPMII: step by step.

Model order	PPMD	+ II	+ UEM	+ SEE1	+ EE1	+ SEE2
2	*2.790*	*2.766*	*2.766*	*2.767*	*2.767*	*2.759*
3	*2.427*	*2.387*	*2.387*	*2.382*	*2.379*	*2.366*
4	*2.310*	*2.254*	*2.254*	*2.235*	*2.230*	*2.212*
5	*2.290*	*2.215*	*2.211*	*2.185*	*2.178*	*2.158*
6	*2.297*	*2.204*	*2.197*	*2.166*	*2.158*	*2.137*
8	*2.319*	*2.196*	*2.186*	*2.150*	*2.142*	*2.118*
10	*2.339*	*2.195*	*2.184*	*2.143*	*2.136*	*2.111*
16	*2.369*	*2.194*	*2.182*	*2.137*	*2.130*	*2.104*

algorithm. Results of measurement[1] of compression efficiency (average bpb), compression time[2] (seconds) and maximal memory requirements (megabytes) are presented in Table 2. As a demonstration of PPM complexity reduction, the last line contains the results for author's implementation of PPMD algorithm.

The table shows that basic PPMII algorithm provides wide range of opportunities. At small D (2–3), it gives compression efficiency comparable to the one of ZIP or of BZIP2 with faster compression speed and smaller memory requirements than BZIP2 ones. At medium D (4–6), PPMII efficiency is noticeable better than ZIP and BZIP2 ones and time/memory requirements remain moderate. Lastly, at high D (8–16), PPMII outperforms the best of described programs PPMZ2 by all characteristics. cPPMII gives even better efficiency, but its low execution speed is not very promising.

Table 2. Integral characteristics of various compressors.

Model order	PPMII			cPPMII		
	Average bpb	Time, sec.	Memory, MB	Average bpb	Time, sec.	Memory, MB
2	*2.759*	*3.18*	*0.6*	*2.716*	*8.51*	*1.1*
3	*2.366*	*3.79*	*1.0*	*2.321*	*10.60*	*1.5*
4	*2.212*	*4.51*	*1.9*	*2.170*	*12.46*	*2.4*
5	*2.158*	*5.21*	*3.5*	*2.114*	*14.00*	*4.0*
6	*2.137*	*5.88*	*5.6*	*2.090*	*15.21*	*6.1*
8	*2.118*	*6.76*	*10.1*	*2.067*	*16.85*	*10.6*
10	*2.111*	*7.25*	*13.3*	*2.057*	*17.57*	*13.8*
16	*2.104*	*7.74*	*16.2*	*2.047*	*18.56*	*16.7*
For comparison						
ZIP -9	*2.693*	*5.93*	*0.5*			
BZIP2 -8	*2.368*	*5.87*	*6.0*			
PPMZ2	*2.139*	not tested	> 100			
PPMD-5	*2.290*	*4.67*	*3.5*			

3. Compression efficiency. In the last experiment, the more detailed comparison is performed for proposed algorithms and for algorithms described in the literature. Following compression schemes are presented in Table 3: the implementation [13] of CTW method [14] with binary decomposition (results were taken from [15]), the associative coder (ACB) by G.Buyanovsky [16], the best of S.Bunton's FSMX coders [8], PPMZ2 by C.Bloom [12]. Next column contains PPMII results

[1]All programs (excluding PPMZ2) were compiled with the same compiler (Intel C v. 4.0), at the same compilation options. The experiments were performed on PII-233 (overclocked up to 292) MHz computer with 64 MB RAM under OS MS Windows98 and FAT32 file system.

[2]PPMZ2 execution time was not measured due to memory insufficiency.

Table 3. Compression efficiency for various compression schemes.

File	CTW	ACB	FSMX	PPMZ2	PPMII-8	cPPMII-8	cPPMII-16	cPPMII-64
bib	*1.782*	*1.935*	*1.786*	*1.717*	*1.732*	*1.694*	*1.679*	***1.676***
book1	*2.158*	*2.317*	*2.184*	*2.195*	*2.192*	*2.136*	***2.135***	***2.135***
book2	*1.869*	*1.936*	*1.862*	*1.843*	*1.838*	*1.795*	***1.782***	***1.782***
geo	*4.608*	*4.555*	*4.458*	*4.576*	*4.349*	*4.163*	*4.159*	***4.158***
news	*2.322*	*2.317*	*2.285*	*2.205*	*2.205*	*2.160*	*2.142*	***2.137***
obj1	*3.814*	*3.498*	*3.678*	*3.661*	*3.536*	*3.507*	***3.497***	*3.498*
obj2	*2.473*	*2.201*	*2.283*	*2.241*	*2.206*	*2.154*	*2.118*	***2.110***
paper1	*2.247*	*2.343*	*2.250*	*2.214*	*2.194*	*2.152*	*2.144*	***2.142***
paper2	*2.190*	*2.337*	*2.213*	*2.184*	*2.181*	*2.130*	***2.124***	***2.124***
pic	*0.800*	*0.745*	*0.781*	*0.751*	*0.757*	*0.721*	*0.715*	***0.704***
progc	*2.330*	*2.332*	*2.291*	*2.257*	*2.215*	*2.178*	***2.161***	***2.161***
progl	*1.595*	*1.505*	*1.545*	*1.445*	*1.470*	*1.433*	*1.398*	***1.390***
progp	*1.636*	*1.502*	*1.531*	*1.448*	*1.522*	*1.489*	*1.414*	***1.391***
trans	*1.394*	*1.293*	*1.325*	*1.214*	*1.257*	*1.228*	*1.186*	***1.172***
Average	*2.230*	*2.201*	*2.177*	*2.139*	*2.118*	*2.067*	*2.047*	***2.041***

for $D = 8$ and last three columns contain cPPMII results for $D = 8, 16, 64$. It is necessary to emphasize the CTW implementation [13] uses symbol decomposition especially optimized for English texts.

As opposed to the other PPM schemes, PPMII works well enough for nontextual data (geo, obj1, obj2). Now, it is really — *universal data compression* ;-).

References.

1. Shkarin, D. (2001) *Improving the efficiency of PPM algorithm.* Problems of Information Transmission, 34(3):44-54, in Russian.
2. Cleary, J.G. and Witten, I.H. (1984) *Data compression using adaptive coding and partial string matching.* IEEE Trans. on Comm., 32(4):396-402.
3. Moffat, A. (1990) *Implementing the PPM data compression scheme.* IEEE Trans. on Comm., 8(11):1917-1921.
4. Howard, P.G. (1993) *The Design and Analysis of Efficient Lossless Data Compression Systems.* PhD thesis, Brown University.
5. Bloom, C. (1998) *Solving the Problems of Context Modeling.* www.cbloom.com/papers/.
6. Shkarin, D. (1999) *BMF — lossless image compressor.* ftp.elf.stuba.sk/pub/pc/pack/bmf_1_10.zip
7. Shkarin, D. (2001) *PPMd — fast PPM compressor for textual data.* ftp.elf.stuba.sk/pub/pc/pack/ppmdh.rar.
8. Bunton, S. (1996) *On-Line Stochastic Processes in Data Compression.* PhD thesis, University of Washington.
9. Martin, G.N.N. (1979) *Range encoding: an algorithm for removing redundancy from a digitised message.* Presented to The Video & Data Recording conference. Southampton.
10. InfoZIP Group (1999) *Zip v.2.3 — compression and file packaging utility.* www.cdrom.com/pub/infozip/.
11. Seward, J. (2000) *BZip2 v.1.0 — block-sorting file compressor.* www.muraroa.demon.co.uk/.
12. Bloom, C. (1999) *PPMZ2 — High Compression Markov Predictive Coder.* www.cbloom.com/src/.
13. Volf, P.A.J. (1996) *Text compression methods based on context weighting.* Technical report, Stan Ackermans Institute, Eindhoven University of Technology.
14. Willems, F., Shtarkov, Y. and Tjalkens, T. (1995) *The context-tree weighting method: Basic properties.* IEEE Trans. on Inf. Theory, 41(3):653-664.
15. Volf P.A.J and Willems F.M.J. (1998) *Switching between two universal source coding algorithms.* Proc. IEEE Data Compression Conf. pp.491-500.
16. Buyanovsky, G. (1994) *Associative coding.* The Monitor, 8:10-19, in Russian.