One attempt of a compression algorithm using the BWT

Bernhard Balkenhol[*]
Fakultät für Mathematik
Universität Bielefeld
Postfach 100 131
33501 Bielefeld
Germany

Yuri M. Shtarkov[†]
Institute for Problems on
Information Transmission
100 447, Moscow, GSP-4
B. Karetnyi per. 19
Russia

1 Introduction

In 1994 Burrows and Wheeler [5] described a universal data compression algorithm (BW-algorithm, for short) which achieved compression rates that were close to the best known compression rates. Due to it's simplicity, the algorithm can be implemented with relatively low complexity. Fenwick [8] described ideas to improve the efficiency (i.e. the compression rate) and complexity of the BW-algorithm. He also discusses relationships of the algorithm with other compression methods. Schindler [14] proposed a Burrows and Wheeler Transformation (BWT, for short) that is based on a limited ordering. This speeds up the algorithm for compression, but slows it down for decompression and slightly decreases the efficiency. Larsson [10] describes the relationship of the BWT with suffix and context trees. Sadakane [13] suggests a method to compute the BWT faster, and compares it to other methods. Recently Balkenhol and Kurtz [2] gave a thorough analysis of the BWT from an information theoretic point of view. They described implementation techniques for data compression algorithms based on the BWT, and developed a program with a better compression rate. In [4] these previous results on the BW-algorithm are improved. Based on the context tree model, the authors consider the specific statistical properties of the data at the output of the BWT. They describe six important properties. These considerations lead to modifications of the coding method, which in turn improve the coding efficiency. Further improvements related to the sorting are presented in [3, 11, 12].

2 Context Tree Models of Sources

Let A be the discrete alphabet of α symbols, $\alpha \geq 2$; x^k be the first k symbols of the message x^n with $x_i \in A$; $p(x^k|\omega)$ be the probability of the occurrence of x^k at the output of the source ω and $\varphi = \{\varphi(x^n),\ x^n \in A^n\}$ be a uniquely decodable (arithmetic) code for sequences of arbitrary (in particular, unknown beforehand) length n. The codeword lengths satisfy the inequality $|\varphi(x^n)| = -\log q(x^n|\varphi) \leq$

[*]Email: bernhard@mathematik.uni-bielefeld.de

[†]Email: shtarkov@iitp.ru, Work done while visiting the Fakultät für Mathematik, Universität Bielefeld

$-\sum_{k=1}^{n} \log \vartheta(x_k|x^{k-1},\omega) + 1$, where $|z|$ is the length of the sequence z or the cardinality of the set z, $\log(.) = \log_2(.)$, $\{q(x^n)\}$ is a coding probability distribution, which is described by conditional probabilities $\{\vartheta(x_k|x^{k-1},\omega),\, x^k \in A^k, k = 1,2,\ldots\}$. The choice $q(x^n) = p(x^n|\omega)$ is the optimal one. But if the statistic of the source is unknown, then the universal coding approach for a given source model or for some set of source models can be used. Below we shall refer to the important set of *Context Tree* (CT) models.

A finite memory CT-source ω is defined by a proper and complete set S of contexts (sequences over the alphabet A) of length $d = |s| \le D$, by the set of conditional probability distributions $\{\Theta_s, s \in S\} = \{\{\theta(a|s), a \in A\}, s \in S\}$, and by the probability distribution for the first D symbols of the message. Completeness and properness of a set S mean that exactly one context exists for any $x^k \in A^k, k \ge D$, i.e. the equality $x_k \ldots x_{k-d+1} = s_k \in S$ holds for exactly one $d \le D$. Then the probability $p(x^n|\omega)$, divided by the probability of the first D symbols, is equal to the product of the conditional probabilities $\theta(x_{k+1}|s_k)$, $k \ge D$, or (in other words) to the product of the probabilities $p(x^k(s)|\Theta_s)$ over all $s \in S$, where $x^k(s)$ is the subsequence of *independent* symbols from x^k occurring after context s. Thus S is a model of the CT-source which can be represented as a set of leaves (contexts) of a complete and proper α-ary tree T_S (and vice versa).

Any CT-source can be described by a Markov chain of order $D' \ge D$ with a *larger number* K of "free" parameters (values of conditional probabilities). Since the cumulative (per message) redundancy of universal (relative to the values of free parameters) coding is proportional to $K \log n$, decreasing K is important for decreasing the coding redundancy under the condition that *it does not decrease* the accuracy of the source description. CT-models satisfy the last condition: their structure allows to exploit the fact that the "actual" lengths of the contexts are *different.*

Usually the number of different symbols that occur after a context, decreases with the length of the context. Consider for example english text: if x_k is a blank and x_{k-1} is a period, then with high probability the next symbol will be *any* capital letter independent of x_{k-2}, x_{k-3}, etc. (the actual length of the context is two). Of course, exceptions exist: for example, if $x_k = q$, then almost surely $x_{k+1} = u$ (although the actual length of the context q is one).

Thus some paradoxical situations occur: the shorter the contexts, the smaller the number K of free parameters and the coding redundancy (see above), and the larger the uncertainty relative to the next symbol and the coding rate. In fact, only rather short contexts influence the coding rate, and increasing D above some threshold does not improve the coding rate. Therefore we shall use conventional terms "bad" and "good" for short and long contexts, respectively, although contexts of almost all lengths exist, as well as exceptions of the kind mentioned above.

3 The BW-algorithm and CT-Models

The BW-algorithm [5] consists of three phases:

1. The Burrows and Wheeler transformation (BWT) $x^n \to y^n$ is the reordering

(permutation) of the symbols of x^n according to the following rule. All n cyclic shifts of x^n (including x^n) are lexicographically ordered, and the sequence of the last symbols of the ordered shifts is y^n. It is easy to show that this is a one-to-one mapping if the position of x^n in the sequence of shifts is described. For more details on the transformation, we refer the reader to [5].

2. Move-to-Front transformation (MTF) $y^n \rightarrow z^n$ of the sequence y^n into the sequence of numbers $z^n = z_1 \dots z_n$, $0 \leq z_i < \alpha$ in the following way: If $y_{k+1} = a \in A$ and $z_k(a)$ is the number for a after k steps, then $z_{k+1}(a) = 0$ and $z_{k+1}(b) = z_k(b) + 1$ for all symbols b with $z_k(b) < z_k(a)$. For any given initial numbering of symbols (known to coder and decoder) MTF is a one-to-one mapping.

3. Noiseless coding of z^n (including run length coding of runs of zeros).

Burrows and Wheeler later also appended a sentinel character (a symbol larger than all symbols occurring in x^n) to x^n, recognizing the fact that it is more efficient to sort non-empty suffixes than cyclic shifts. However, there is another reason for appending the sentinel: it prevents from introducing dependencies between parts of the input sequence which are actually not present. But as in [2] we use the sentinel only as a virtual symbol for sorting but not for the description of the starting position of the original sequence in the transformed sequence. If we apply the (modified) BWT to the reverse $x_n \dots x_1$ of x^n, followed by the sentinel, then the first k symbols of it's periodical shift $x_k \dots x_1$ *sentinel* $x_n \dots x_{k+1}$ form the *context of maximal length* for x_{k+1}, i.e. for the last element of this shift (ignoring symbols $x_n \dots x_{k+2}$). Therefore the BWT of any message corresponds to the lexicographically ordered (not necessarily complete) α-ary tree T^*, describing the ordered set of contexts, and y_i is the only symbol generated in the ith leaf of T^*. Then the connection of the BW-algorithm and PPM*, as mentioned in [8], is clear. And the limited ordering, proposed in [14], corresponds to the context tree $T(D)$ for a Markov chain of order D, $y^n = x^n(s_1), x^n(s_2), \dots$ (see Section 2) and such a modified BW-algorithm corresponds to PPM [7] (see also [8]).

3.1 Properties

As any one-to-one mapping, the BWT does not change the entropy of the source or the probability of the message, but the (relatively) low complexity of the BWT makes it very attractive for data compression programs. Therefore it is important to look for an efficient coding of y^n, taking into account the difference of the probability distributions for x^n and y^n over $\mathcal{X}^n$. But for studying the properties of y^n, we have to know at least some properties of x^n. The following properties of y^n correspond to the CT-model of x^n which is consistent with real input sequences (see Section 2).
The first (lets say t) symbols of x^n may have no context in S because the length of the past is to small. Therefore we have to ignore the corresponding positions in y^n and the position of the sentinel in y^n during the description of the properties.
Let $s_i \in S$ be the context of symbol x_i and let $c_1, \dots, c_r$ be the elements of S in lexicographic order.

$$Pr(x_1,\ldots,x_n) \;=\; Pr(x_1,\ldots,x_t)\prod_{i=t+1}^{n} Pr(x_i|x_1,\ldots,x_{i-1})$$

From the Definition of a CT Source it follows:

$$= \;\; Pr(x_1,\ldots,x_t)\prod_{i=t+1}^{n} Pr(x_i|s_i)$$

We have divided our sequence into an independent one such that we can take first a subsequence of the symbols in x^n having the same context without changing the probability:

$$= \;\; Pr(x_1,\ldots,x_t)\prod_{s\in S}\prod_{i:s_i=s} Pr(x_i|s)$$

Thus the model S define only the dividing of the sequence y^n into *fragments* $y^n(s)$. Any fragment $y^n(s), s\in S$ (as well as $x^n(s)$) is the sequence of independent, equally distributed symbols from $\mathcal{X}(s)$, where $\mathcal{X}(s)\subset\mathcal{X}$ is the subset of symbols of the alphabet, occurring in $x^n(s)$, S is the *real* but unknown Context Tree (CT) model, and the corresponding (conditional) probability distributions $\theta(a|s), a\in\mathcal{X}$ depend on s. Obviously we can take these subsequences $x^n(s)$ such that the contexts are in lexicographic order:

$$= \;\; Pr(x_1,\ldots,x_t)\prod_{j=1}^{r}\prod_{i:s_i=c_j} Pr(x_i|c_j)$$

Again due to the independency we do not change the probability if we define a permutation π_j of the subsequence of x^n for a given context c_j

$$= \;\; Pr(x_1,\ldots,x_t)\prod_{j=1}^{r}\prod_{i:s_i=c_j} Pr(x_{\pi_j}(i)|c_j)$$

Of course one can find permutations $\pi_1,\ldots,\pi_r$ such that:

$$= \;\; Pr(y_1,\ldots,y_n).$$

Property 1: y^n is the sequence of *independent* symbols over $\mathcal{X}$ with *variable* probabilities of occurrence (the probabilities can be different for different contexts). It corresponds to an *infinite memory* of the source generating y^n, although the original CT-source has a restricted depth of memory.

Property 2: y^n consists of "good" and "bad" fragments corresponding to good and bad contexts, respectively. The probabilities of occurrence of symbols do *almost* not change inside the same fragment (in fact, it can be an undetectable concatenation of "close" fragments), but it can change essentially between two fragments.

Property 3: The statistics of the fragments (i.e. the sets of different symbols in the fragments) are different.

Property 4: The number of different symbols in a fragment usually decreases with the actual length of the corresponding context (in particular, most of the good fragments consist of repetitions of one symbol and this is one of the reasons for using MTF and run length coding as it was originally proposed in [5]). Therefore the method of *multi-alphabet* coding, which allows to adapt to an unknown subset of symbols in the fragments, should be used; MTF and grouping of numbers, as i.e. proposed in [8], define such a coding method.

Property 5: The longer the common prefix of two contexts (the "closer" they are), the smaller the difference of the sets of symbols generated at any of these contexts. This is one more reason for applying MTF (and, in fact, this is the basis of PPM).

Property 6: With an increasing message length at the output of any given CT-source, the number of fragments slightly increases (because for a short message length some subsequences $x^n(s)$ (see Section 2) are empty). Thus the (average) length of the fragments grows almost linearly with the growing message length.

Usually during sequential universal coding, the statistics of only the previous part of the message is used (it is known that the pre-coding description of the statistics of the message does not give us an advantage). Therefore the continuous jumping from one fragment to another one (typical for context-based coding) does not decrease the efficiency of coding. And knowledge of the "current" context can be used very efficiently (see, for example, PPM). Thus the BWT does not give us some additional advantages (for increasing efficiency) but results in the loss of knowledge about the context of the coded symbol. So we may suppose that *the efficiency of the BW-algorithm can not reach the efficiency of the best context-based algorithms* (such supposition was formulated in [8] as well). Nevertheless because the best context based algorithms require considerably more space and time than the BW-algorithm, it is desirable to increase the efficiency of the latter as much as possible.

4 The updating of the statistics

Due to the properties of a CT model for describing the next symbol x_{k+1} of x^n we should use only the set of frequencies of the symbols with the same context than x_{k+1}. It seems that the same approach can be used for the coding of y^n, since at most cases the permutation of the symbols of $x^n(s)$ does not influent the results of coding. In this case it is sufficient to keep only the set of frequencies for the "current" fragment s and drop all the frequencies to zero after the end of this fragment, i.e. in the beginning of the next one (this is the optimal updating strategy). But the

problem is that we can not divide the current part y^k of y^n into fragments without *outside information* even for known model S. Furthermore S is also unknown.
It is possible to propose coding algorithms, ignoring this problem entirely (they do not use the "fragmentation"). In fact all the known methods correspond to such an approach since they use the frequencies of *all the encoded part of the file* without any attempts of its' updating (except the strategy described in [2]); then implementation of MTF allows to avoid the dramatical increasing of the coding rates. Nevertheless the efficiency of simple updating algorithms has to be studied for this approach as well.
But the most promising approaches to increasing the coding efficiency are connected with the concept of fragmentation. In this case the group of coding problems is connected with unknown fragment bounds and unknown specification (labeling) of fragments (which context s correspond to the current fragment and which to the previous ones). In other words, at any step of coding we wish to know answers on three questions:

1. Does x_{k+1} belong to the same fragment as x_k (then we can use the frequencies of the current fragment) or not ?
2. How to define the frequency set for coding or (in other words) where is the beginning of the current fragment ?
3. How to find the other frequency sets which can be useful for coding (for example, for describing of new symbols, as in PPM) or how to find contexts, corresponding to different fragments (problem of fragment specification) ?

4.1 Updating rules without fragmentation

Since the (average) length of fragment is approximately proportional to n, we can propose simple and clear updating rule.

Proposal 1. *Arbitrary updating*

$$\tau_k(z) \to F(\tau_k(z)) \tag{1}$$

is made, when the current length of already encoded part of z^n is equal to

$$k = k_\nu = \nu(\xi n), \tag{2}$$

where ξ, $0 < \xi < 1$, is a parameter and ν, $1 \leq \nu < 1/\xi$, is an integer (it is convenient to substitute ξn in (2) by it's integer part).

Now we must define the left side of (1). According to the properties of the CT model it is natural to use $F[\tau_k(z)] = 0$. But the moments k_ν usually do not coincide with the bounds of fragments. Besides, every updating interval of size ξn can contain several fragments, the part of one fragment, the bound between fragments, etc. Finally, this rule ignores the *apriory* knowledge of the fact that usually

$$Pr(z_1) > Pr(z_2), \qquad \text{if} \quad z_1 < z_2. \tag{3}$$

Therefore it is more reasonable to use less "radical" updating rules; for example

$$F(\tau(z)) = \tau^*(z), \tag{4}$$

where $\tau^*(z)$ are **independent** of $\tau(z)$ (for example, $\tau^*(0) = 5$, $\tau^*(1) = 3$, $\tau^*(2) = 2$, $\tau^*(3) = \tau^*(4) = 1$ and $\tau^*(z) = 0, z > 4$), or

$$F(\tau) = \tau/\mu, \tag{5}$$

where $\mu > 1$ is a real value.

It is reasonable to choose the values of ξ and μ, which minimize the average coding rate for *all* files of Calgary Corpus (let us note that this is optimization over 2 parameters *simultaneously*).
The same approach can be used for the rule (4) as well. But the more number of "free" parameters the more artificial is this approach.
The similar updating can be used in the process of *direct* coding of y^n (without MTF).

An other approach for updating is the estimation of moments when the statistics of y^k changes essentially. It is tightly connected with adaptive fragmentation (see Sec. 4.4).

4.2 Description of "superfragment" bounds

It was mentioned that the moments of updating do not correspond to the bounds of fragments in the both above considered methods. This disadvantage can be weakened by the *direct description* of *some* fragment bounds.
Let $k(a)$ be the first symbol in y^n, corresponding to context with the first letter a ($k(a_1) = 1$). Then all the symbols y_j with j, satisfying inequality

$$k(a_i) \le j < k(a_{i+1}), \tag{6}$$

correspond to all contexts (and fragments) with a_i as the first symbol. In other words, all the symbols y_j, satisfying (6), form "superfragment" of length $n(a_i) = k(a_{i+1}) - k(a_i)$.
Proposal 2. *Describe all $k(a)$, $a \in \mathcal{X}_0$, and make updatings of all $\tau_k(z)$ at all the moments*

$$k = k(a) - 1, \qquad a \ne a_1. \tag{7}$$

If n and m are known then for description of bounds of "1-symbol" superfragments we must spend

$$l(B) = \log_2 \binom{n-1}{m-1} \approx (n-1)H[(m-1)/(n-1)] - 0.5 \log_2(n-1) \tag{8}$$

bits, since $k(a_1) = 1$.

file	*length*	M	*depth 1*	*depth 2*
bib	111261	81	0.009	0.159
book1	768771	81	0.001	0.028
book2	610856	96	0.002	0.059
geo	102400	256	0.029	1.346
news	377109	98	0.003	0.112
obj1	21504	256	0.112	4.947
obj2	246814	256	0.013	0.677
paper1	53161	95	0.020	0.358
paper2	82199	91	0.013	0.206
pic	513216	159	0.003	0.085
progc	39611	92	0.027	0.501
progl	71646	87	0.014	0.217
progp	49379	89	0.021	0.368
trans	93695	99	0.012	0.280
	3141622		0.020	0.667

Table 1: Superfragments (in bits/byte) for the Calgary Corpus

- The increasing of coding rate due to description of $\{k(a), a \in A_0\}$ is very small (for $m = 100$ and $n = 50000$ it is less than $H(0.002) = 0.021$ bits/symbol; this additional rate decreases, when n increases);

- The changing of the first symbol of context *always* means the essential changing of statistical properties of fragments. Therefore the updating of frequencies in the moments (7) are absolutely reasonable. In spite of we must update the statistics in the beginning of any fragment, the rule (7) let us increase the number $m-1$ of updatings *in the right moments* (compare with Proposal 1). In our example it is equal to $m - 1 = 99$!

- With the help of $\{k(a), a \in \mathcal{X}_0\}$ we can define all

$$\tilde{t}_n(a_i) = k(a_{i+1}) - k(a_i), \qquad i \leq m, \tag{9}$$

  where $k(a_{m+1}) = n+1$ (in fact, this property is used at the first step of the BW decoding). The equalities (8) are useful for coding y^n but not z^n !

Of course, we can define superfragments by *two* first symbols of contexts. The cost of its' description increases essentially (around 0.1 bit/symbol) but this smaller superfragments are more uniform statistically. Beside, in this case the analog of (9) gives us information about frequencies of *couples* of symbols and about $\tau_n(z)$. Furthermore the calculations of bounds (mentioned above) of some smaller superfragments will give us more information. With the frequencies of all pairs of symbols we know for each context of depth 1 the sub-alphabet and the frequencies of the symbols occurring with this context. Finally, the coding procedure can be made more flexible and better corresponding to CT model.

The most natural updating rule for y^n is $F(\tilde{t}) = 0$ when k satisfies (7). And we can use (4) or (5) for z^n. The experimental data (in bits/byte) for describing the superfragment bounds of the depths 1 and 2 for the files of the Calgary Corpus are presented in Table 1.

4.3 Nonuniform superfragments

At most cases the contexts, starting with “non-letter” symbols or “letter” ones, have very different statistical properties. In the second case the conditional probability distributions are usually very “degradated” (contain a few nonzero values, and one of this values is close to 1). In the first case such distributions contain many nonzero values with more uniform probability distribution. In fact, the compression effect mostly arises due to efficient coding of subsequences (fragments) of the second type. Thus, if we need to restrict number of groups of contexts (for restriction of the description length) then it is reasonable to start with grouping of contexts of the first type. The same is true for grouping of fragments of y^n (it is well illustrated by printed sequence y^n for “Paper1”, where the first part, corresponding to contexts of the first type, is very “noisy”). Therefore the superfragmentation of y^n must be more detailed for the fragments of the second type.
The last recommendation can be used if “non-letter” and “letter” symbols are ordered in alphabet $\mathcal{X}$ in such a way that *two rather compact groups* of such symbols take a place. From this point of view ASCII code is close to optimal (but some small reordering of symbols can be useful see section 5.1).
At any case we can consider the following versions of nonuniform superfragmentation.

1. Two superfragments (only !) are defined by the bound, corresponding to the beginning of the first “letter” fragment.

   This simple and “cheap” method let us divide y^n into two parts with very different statistics and two different methods can be used for coding this parts of z^n or y^n.

2. In addition to the previous case we divide the second part of z^n (y^n) into superfragments, corresponding to different first “letter” symbols of context (this is the partition, considered in Sec.4.2, but for the second part of z^n (y^n) only).

3. Uniform fragmentation of Sec.4.2 (as a particular case).

4. Dividing the second (of two) superfragment into superfragments, corresponding to two first “letter” symbols of contexts (the first superfragment does not divided into the parts or divided into superfragments, corresponding to first “non-letter” symbol of contexts).

5. Uniform fragmentation of Sec.4.2 for the first two symbols of the contexts.

This particular cases are ordered according to description length of bounds of superfragments. Beside, the partition becomes more and more exact.

4.4 Adaptive fragmentation

As we mentioned above, the description of fragments’ bounds is very “expensive”. In spite of it exists some possibility to calculate the bounds of some fragments with the help of the set of superfragments’ bounds (see Sec.4.2) there are many cases when

we can use only *adaptive fragmentation*, based on the analysis of the already encoded part of y^n (or z^n).
Most of coding methods use the current frequency set of the current fragment. It means that we need to estimate the left-hand side bound of the current fragment. But which criterion we should use for estimation ? There is only one reasonable answer: *coding efficiency*. This approach results in the following properties.
1) The estimate of the left-hand side bound of the current fragment *can change* with appearance of new symbols. It changes the frequency set for coding of the *next* symbols (but we can not change something for the symbols, which are already encoded).
2) Two or more *real* neighbor fragments with repetitions of the same symbol will be unavoidably estimated (considered) as one fragment.

The second property is positive; it looks like the generalizing of PPM*. Therefore the adaptive fragmentation corresponds to *the set of models which is wider than the set of CT models* (with the same maximal depth of contexts). In fact, if sequence w^i is an internal node of the context tree then it must be continued for all previous symbols $a \in \mathcal{X}_0$. In our case some sequences $\{w^i a_j,\ w^i a_{j+1}, \cdots\}$ can be considered together but independently of another continuations of w^i. This is another generalization of CT model than in [16]. It's efficiency strongly depends on the chosen ordering of the alphabet symbols since we can combine only the neighbor fragments.
It is not easy to transform the criterion of "coding efficency" into the estimation rules.

But what is the reason to describe bounds of superfragments if we use adaptive fragmentation ? There are several answers on this question.

- We can use the fragmentation rules, depending on corresponding superfragments (it is useful to develop at least two different rules for the two parts of y^n or z^n).
- Any adaptation let us define the bounds (and the number !) of fragments with some error, and the exact knowledge of superfragment bounds is useful (for coding).
- The superfragmentation let us define the bounds of some fragments (see Sec.4.2).

Thus both approaches have to be used together.

Now let us consider the possibility of implementation of the main property of PPM (as we know, it is very efficient). The concept of PPM can be described in the following way.
The sets of frequencies $\{t_k(a|w^j), a \in A(w^j)\}$, $j = D, \ldots, i$, are used for coding y_{k+1}, where $w^j = x_k, \cdots, x_{k-j+1}$ and i is the *maximal* integer such that $y_k \in A(w^i)$ (we are not going into details). Therefore it is necessary and sufficient to know all the fragments of y^n, corresponding to different contexts of length D. We know only the bounds of superfragments, which were described by $l(B)$ bits (see, for example, (8)) and can calculate the bounds of some fragments and label them properly (see Sec.4.2).

But it is not sufficient; in all the other cases the sequential adaptive fragmentation does not let us to label fragments. So we need to modify PPM for y^n.
In fact, PPM is based on the assumption of *closeness of conditional probability distributions* and the normalized frequency sets for "close" contexts (fragments) (we are not going into details here). For y^n this property can be reformulated in the following way. Let $h(a)$ be the last (maximal) position of symbol a in the already encoded part y^k of y^n. The less the difference $k - h(a)$ the more probability that $x_{k+1} = a$. This qualitative description of the "closeness concept" must be formulated in some quantitative form.
For example, we can use "the coding conditional probabilities"

$$\vartheta(a) = \frac{1}{c}\, g\left(\frac{k - h(a)}{n}\right) \tag{10}$$

where $g(.)$ is the decreasing function of it's argument and c is the normalizing factor (sum of values of function $g(.)$ over all $a \in \mathcal{X}_0$). Dividing by n in the argument of $g(.)$ let us take into account that we should use the distance *relative to the average fragment length* or, in simplified form, to message length n - compare with Sec.4.1. But if the superfragments are described then it is better substitute n by the length $n(a)$ of the current superfragment.
If we have no sufficient information for the choice of function $g(.)$ then we can use only the *ordering* of symbols according to the value of $k - h(a)$ and one of standard coding methods (for example, Levenstein code or arithmetic coding). Thus we returned back to general concept of MTF (with rather clear argumentation) and it's efficiency becomes more clear as well. And if we'll take into account that in PPM we consider *separately* the *groups* of symbols, corresponding to different lengths of context (see above), we should divide the ordered sequence of symbols into groups, corresponding to current fragment and to superfragments of "different levels"; in the simplified form it corresponds to MTF.
It is clear that the sizes of groups must be *variable* and not fixed. Since we do not know the bounds and labels of all the fragments, we can introduce the variable grouping with the help of the set of *distance thresholds*

$$R_i = \chi_i n_c, \qquad i = 1, \dots q, \tag{11}$$

where $q + 1$ is a number of groups, n_c is a length of current superfragment and $\{\chi_i\}$ is the set of parameters (to be optimized). If symbol a satisfies the condition

$$R_{i-1} \le k - h(a) < R_i, \tag{12}$$

then it belongs to i-th group ($R_0 = 0$ by definition). Such approach is better matched with PPM but not reliable (due to shortage of information).

With such approach the results of adaptive fragmentation are defined only by *the concept of the minimum description length* for any *given coding method* (it can influence the results of fragmentation essentially).

4.5 Conclusion for the updating

The consideration above let us come to the following conclusions:

- This is the attempt to use the statistical model of y^N for extracting some features of the efficient coding in the way, *close to logical*;
- This consideration let us understand some connection of the known coding methods (as PPM) and of the known BW coding methods after MTF. Simultaneously it defines some directions of increasing of the coding efficiency;
- According to DCC-98 (p.499) the average coding rate of PPMD for 12 files of Calgary Corpus is equal to 2.284. If we'll correct the strange value 2.160 for book1 (it must be 2.294) then we'll receive the average rate 2.294, i.e. only 0.006 bits/symbol better than our result in DCC-99 (!). On our opinion it can be explained only by the fact that the coding of y^n corresponds to the more wide class of source models than the class of CT models (see above in Sec.4.4). It means that we may hope to receive *better results than* PPMD.

Choosing the coding algorithm we must take the following into account.

Remark 1 There are many "short" fragments in y^n (especially for short files). As a rule the separate coding of the short fragments is **useless** since the statistics of fragment can not appear at the short sequence. Therefore it is reasonable for coding *to combine* the short neighbor fragments.

Remark 2 The maximal dependence of the conditional probability distribution on w^i is maximal, when $w^i = s \in S$. But this important fact is used in known algorithms in the small degree, since:

a) In particular, MTF is equivalent to supposition that the probability of appearance of symbol a after symbol a in y^n is independent of a. This is a very rough approximation.

b) Furthermore MTF and the application of all the "current" statistics of z^k exclude the possibility to take into account the dependence of conditional probabilities on context as well. And this is also not true.

It is possible to remove the disadvantage a) *only* by refuse from MTF. Therefore the coding of y^n (not z^n) is more preferable.
It is possible to remove the disadvantage b) *only* with the help of the proper *updatings*. But it follows from the Remark 1 that the defining of moments of updatings is not a very clear problem.

4.6 The problems of direct coding of y^n

The proper use of (updated) frequencies of appearance of numbers [2, 4] let us decrease the coding rate of z^n. But the frequency of any number z is some mixture of frequencies of different symbols with *different* probabilities of appearance. Furthermore this probabilities depend on the position of the symbol inside z^n (y^n). Therefore the further decreasing of the coding rate is related in first turn with the transition from coding of numbers (z^n) to coding of symbols (y^n).
Below any fragment with the same probability distribution of symbols will be named *uniform* fragment; it corresponds to a subsequence of symbols, generated in some state (context) of the Context Tree source. Then any observed fragment is the concatenation of uniform fragments.

4.6.1 The generalized frequencies (updating problem)

At any current moment (after coding of y^k, $k < n$) we do not know the bounds of uniform fragments due the properties of the BWT and the *unknown* Context Tree model of source. But according to Property 5 (see above) the statistical *closeness* of two uniform fragments of y^n depends on *distance* between them; the less the distance the more the closeness (at average) and vice versa. It seems that this is not the appearance of non-stationarity; such y^n can be described by a stationary switching model [15].
Thus our "believe" to any previous symbol should decrease with the increasing of it's distance from the current end of y^n (y^k). It means that in the expressions for coding conditional probabilities we should use *generalized* frequencies, which depend on frequencies itself and their distribution in the file (their distances from y_k).
In fact, the original MTF coding corresponds to entire ignoring of frequencies and to strong dependence on the distances; the rule $z_k \to 0$, where z_k corresponds to the last encoded symbol, means that this symbol will be described in the shortest way just if it appeared the first time. The modified MTF rule $0 \to 0$, $1 \to 0$ and $z \to 1$, $z > 1$, introduces the weakest dependence on frequencies. Developing of this approach bring us to the wide set of different MTF, for example: $0 \to 0$; $z \to 1$, $z > 1$; $1 \to 0$, if privious symbol was different form 0. The last version is used in our algorithm as we will see more detailed in Section 5.2
The *updating* of frequencies is one of the most natural and promising ways of introducing the generalized frequencies. But it can be made by many different methods and the choice of updating method is the important problem (see Section 4).

4.6.2 The "pressure of runs"

Let us define run fragment (or *run*) as repetition of the same symbol $\lambda > 1$ times; any fragment, which does not include runs, is a *noisy fragment*. If λ is small then rather often the uniform fragments include runs; if λ is large then often the noisy fragment is the concatenation of uniform fragments. Therefore the value of λ must be the result of compromise; we choose $\lambda = 4$.

According to this definitions any y^n is the concatenation of *runs* and *noisy fragments* "…– run – noisy fragment – run – noisy fragment – …". It is obvious that just the neighbor run (for example, of symbol a) and noisy fragment have different probability distributions. Therefore generalized frequency of symbol a does not correspond to the probability of a in the noisy fragment. It results in overestimating of coding conditional probability for a, underestimation of coding conditional probabilities for another symbols and decreasing of coding efficiency (this phenomenon is named as "pressure of runs"). The more the length of run the more this negative effect. Any reasonable updating procedure decreases this pressure but does not exclude it entirely. Thus it is another important problem.

4.6.3 Multi-alphabet properties

At most cases the number μ of different symbols in the uniform noisy fragment is essentially less than $|\mathcal{X}'|$. And since the redundancy of universal coding usually proportional to $\mu - 1$ or larger value, it is important to use the coding, at any moment depending on μ (or close value) not $|\mathcal{X}'|$. This is *multi-alphabet* coding: it let us partly compensate the lack of knowledge of the bounds of uniform fragments.

The multi-alphabet properties of the original BWT-MTF are appeared as a result of reasonable *grouping* of numbers (we are not going into details here). But using the *fixed* sizes of groups contradicts to the properties of y^n; the value of μ changes from one uniform noisy fragment to another one in the wide range. Therefore we should look for more flexible methods of multi-alphabet coding, in particular, applying the groups of *variable* (depending on y^k) sizes.

Since we do not know the bounds of uniform fragments, one of the simplest (and rather natural) ways of changing the sizes of groups is based on the applying of the set of *sliding windows* of different sizes. This approach will be considered in the next Section in more details.

Remark. Since for any given CT model of source the (average) length of uniform fragment grows linearly with growing of n the sizes of sliding windows should be the linear function of n. In fact, for small n this growing is slowlier (in the beginning of file the best CT model and the corresponding number of uniform fragments grow also). The choice of dependence of sizes of sliding windows on n is important and interesting problem.

The described problems influented our choice of coding algorithm, which is described in the next Section.

4.7 Mixed number-symbol coding algorithm

There are two arguments for considering the mixed algorithm.

- The specific of symbol coding (see previous Section) demonstrates that the transition from any BWT-MTF algorithm (for z^n) directly to symbol coding algorithm (for y^n) will be accompanied by many unpredictable effects. Therefore it is difficult to analyze the set of such effects and to correct algorithm properly.

- Some advantages of the ternary sequence, which we did not use earlier, will be used now.

Let us come back to the number coding algorithm, described earlier.
Any BWT-MTF transform z^n is described by the ternary sequence z_1^n, which is the result of substitution of any $z \geq 2$ by 2, and by the sequence z_2^n, which contains sequential description of all $z \geq 2$ (i.e. of all $n' \leq n$ numbers “2” in z_2^n). It is easy to check that it is sufficient to know z_1^{k+1} and y^k for the proper choice of conditional probabilities for y_{k+1}.
Such algorithm has following advantages:

1) The different algorithms for coding z_1^n and z_2^n can be chosen and optimized *independently* one from the another one.

2) In typical cases the most of symbols (corresponding to numbers 0 and 1) are described by the simple (ternary) sequence z_1^n, i.e. the simplest multi-alphabet property is introduced by such approach.

3) The sequence z_1^n let us divide z^n into the sequence of runs and noisy fragments (see Sec. 4.6.3). All numbers “2” of z_2^n correspond to noisy fragments.

The first of this properties let us use *number coding* for z_1^n and *symbol coding* for z_2^n; in other words, this is mixed number-symbol algorithm. Let us consider both “sub-algorithms”.

5 Increasing the Efficiency of the BW-algorithm

In this section we show how to exploit the properties from the previous sections to increase the efficiency of the BW-algorithm. The general scheme is related to the approaches described in [4], but there are some important differences, mentioned below.

5.1 Order on the symbols

To use “lexicographic” order of the contexts we implicit assume that we have an order on the symbols. In practice usually the order defined by the ASCII code is used. The main advantage of the sorting is, that symbol with a common context are grouped together. This effect is still valid if we define an other order than the ASCII order on our symbols. More detailed the symbols which occur after an “a” are more similar to that ones after an “e” than that ones after an “b”. Therefore we define as an order on the alphabet

$$aeioubcdgfhrlsmnpqjktwvxyz$$

as the order of the small letters and the same one on the capital letters, respectively. Furthermore we exchange (compared to the ASCII order) the symbols “,” and “-”, “?” and “””, “=” and “>”. The order is close to that presented in [6]. If for different contexts the alphabet is more close, then we can use some knowledge from the last one (see section 4.3).

file	length	$\mid \mathcal{X} \mid$	0 (in %)	1 (in %)	2 (in %)
bib	111261	81	66.781	8.693	24.526
book1	768771	82	49.763	15.326	34.911
book2	610856	96	60.849	12.555	26.596
geo	102400	256	35.263	6.767	57.971
news	377109	98	57.966	10.235	31.799
obj1	21504	256	52.437	5.497	42.067
obj2	246814	256	65.415	7.854	26.731
paper1	53161	95	58.402	11.324	30.274
paper2	82199	91	55.319	12.274	32.407
pic	513216	159	87.395	2.889	9.716
progc	39611	92	60.349	11.156	28.495
progl	71646	87	72.875	8.732	18.393
progp	49379	89	74.009	8.641	17.349
trans	93695	99	79.159	5.964	14.877

Table 2: output of MTF in the ternary sequence for the Calgary Corpus

5.2 Modification of the Move-to-Front Transformation

Without actually knowing the context tree modeling the tree source, the Burrows and Wheeler-Transformation permutes the input sequence in such a way that characters with the same right context are grouped together. Consider the context $s \in S$ and let $\mathcal{X}_s$ denote the set of characters in x^n with this context. Because a context restricts the choice of the characters preceeding it, the size of the set $\mathcal{X}(s)$ is usually small. Of course, $\mathcal{X}(c_j)$ and $\mathcal{X}(c_{j+1})$ may be different. However, since the contexts are in lexicographic order, the difference between $\mathcal{X}(c_j)$ and $\mathcal{X}(c_{j+1})$ is usually not too large, i.e. there is local stability. Unfortunately, we cannot immediately exploit this local stability, since we do not know when the contexts switch. For this reason, we transform the local stability into a global one using a move-to-front transformation. The idea of this transformation is to replace each symbol c by the number of distinct symbols which occurred since the last occurrence of c. Therefore the Move-to-Front Transformation is a mapping $MTF : \mathcal{X} \to \{0, \ldots, \mid \mathcal{X} \mid -1\}$. In [8, 14] different modifications of MTF are discussed.
For example starting with the alphabet $\{a, b, c, d, e, f, g\}$ (in lexicographic order) the MTF of a sequence *aaadaaaadaaaagaaaadaaa* is computed as follows

a	a	a	d	a	a	a	a	d	a	a	a	a	g	a	a	a	a	d	a	a	a
0	0	0	3	1	0	0	0	1	1	0	0	0	5	1	0	0	0	2	1	0	0

Before we start to change the rules for MTF let us concentrate only on the cases $0, 1, \geq 2$ at the output of MTF. Since as described in Section 4.7 we will use MTF only for these numbers. Let us denote the sequence where we replace each number ≥ 2 by a 2 at the output of MTF as *ternary sequence* (see Section 4.7). Then we get as output of the transformation:

file	length	$\mid \mathcal{X} \mid$	0 (in %)	1 (in %)	2 (in %)
bib	111261	81	63.836	11.809	24.354
book1	768771	82	51.458	14.883	33.659
book2	610856	96	60.456	13.621	25.923
geo	102400	256	35.380	7.528	57.092
news	377109	98	54.529	14.050	31.422
obj1	21504	256	49.442	9.063	41.495
obj2	246814	256	60.716	12.716	26.568
paper1	53161	95	54.850	15.071	30.078
paper2	82199	91	53.807	14.241	31.952
pic	513216	159	88.556	2.253	9.191
progc	39611	92	56.073	15.440	28.487
progl	71646	87	68.421	13.355	18.224
progp	49379	89	68.456	13.864	17.680
trans	93695	99	73.154	11.902	14.943

Table 3: occurence of the numbers in the ternary sequence for the Calgary Corpus

a	a	a	d	a	a	a	a	d	a	a	a	a	g	a	a	a	a	d	a	a	a
0	0	0	3	1	0	0	0	1	1	0	0	0	5	1	0	0	0	2	1	0	0
0	0	0	2	1	0	0	0	1	1	0	0	0	2	1	0	0	0	2	1	0	0

The occurence of the numbers are presented in Table 2. Now let us start with the following modified rule: Let us consider the context " the". With high probability in an english text the next symbol will be again a "space". Now there are of cause also more possible words starting with " the", but even less likely than "space". The role of "space" in the example above is taken by the symbol "a". For such a context mostly "a" will occur (typically as runs) and in between there are the other symbols (typically isolated). In other words if a new symbol has occurred we produce one step later a 1 in order to move our most likely symbol back to the head of our ordered set of symbols (the alphabet in the local order). To avoid this we change the rule as follows to give the local symbol of a run a second change: if the next symbol has the current number z, then after its coding we change the number as follows: if $z > 1$ then shift z to position 1 else shift z to position 0. Using this rule we get

a	a	a	d	a	a	a	a	d	a	a	a	a	g	a	a	a	a	d	a	a	a
0	0	0	3	1	0	0	0	1	1	0	0	0	5	1	0	0	0	2	1	0	0
0	0	0	2	1	0	0	0	1	1	0	0	0	2	1	0	0	0	2	1	0	0
0	0	0	2	0	0	0	0	1	1	0	0	0	2	0	0	0	0	2	0	0	0

As in [4] we like to encode first this ternary sequence and after that we have of cause to describe the symbols which are replaced by a 2.
Typically, the frequency of the numbers at the output of the MTF transformation monotonically decreases while the number increases. This is e.g. shown in Figure 1 for the files *book1* and *pic* of the Calgary Corpus.

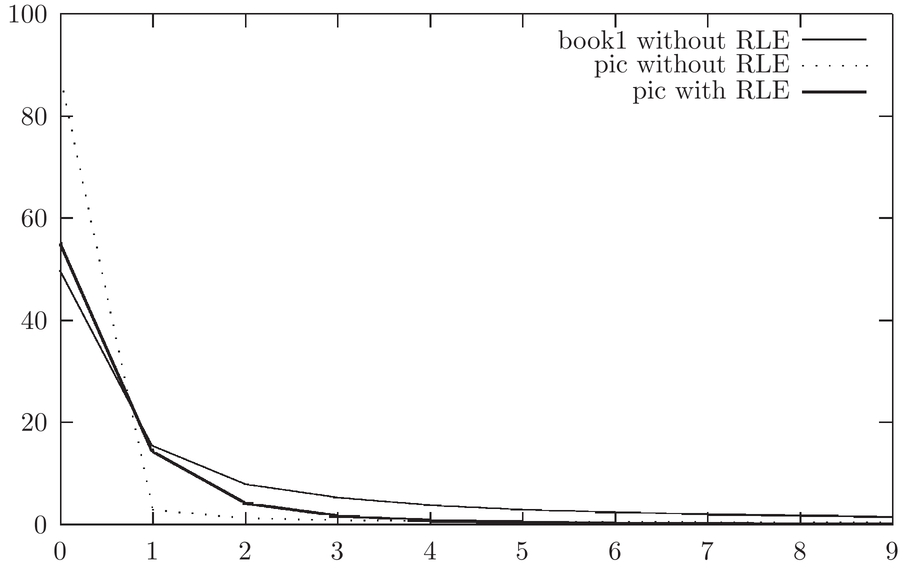

Figure 1: Relative frequencies of characters $[0, 9]$

Therefore most symbols are encoded only with the ternary sequence (see Table 3 for the modified MTF rule).
To increase further the number of zeros we change the rule again: In addition to the privious rule we move a symbol at position 1 only to position 0 if the last produced number was different from zero.

a	a	a	d	a	a	a	a	d	a	a	a	a	g	a	a	a	a	d	a	a	a
0	0	0	3	1	0	0	0	1	1	0	0	0	5	1	0	0	0	2	1	0	0
0	0	0	2	1	0	0	0	1	1	0	0	0	2	1	0	0	0	2	1	0	0
0	0	0	2	0	0	0	0	1	1	0	0	0	2	0	0	0	0	2	0	0	0
0	0	0	2	0	0	0	0	1	0	0	0	0	2	0	0	0	0	2	0	0	0

Notice that we increase the number of zeros in case where the symbol from the privious run will reoccur. In the situation where we have two runs (of differnet symbols) we do not produce any further 1 than that one which we produce allready by the first changing of the MTF rule in case where the second symbol is on a position > 1. Only in case when the second symbol is at position 1 we replace a 0 by a 1 and we loose, i.e.

$$aaaaabbbbbb \quad \rightarrow \quad 00000210000$$

in the first case and

$$aaaaabbbbbb \quad \rightarrow \quad 00000110000$$

in the second one.

file	length	$\mid \mathcal{X} \mid$	0 (in %)	1 (in %)	2 (in %)
bib	111261	81	63.821	11.871	24.308
book1	768771	82	52.841	13.744	33.416
book2	610856	96	61.221	12.990	25.789
geo	102400	256	35.476	7.532	56.992
news	377109	98	54.471	14.133	31.396
obj1	21504	256	49.233	9.142	41.625
obj2	246814	256	60.217	13.236	26.546
paper1	53161	95	54.792	15.160	30.048
paper2	82199	91	54.043	14.073	31.884
pic	513216	159	88.747	2.122	9.130
progc	39611	92	55.684	15.826	28.490
progl	71646	87	67.711	14.038	18.251
progp	49379	89	67.259	15.027	17.714
trans	93695	99	72.517	12.507	14.976

Table 4: The MTF2 rule for the Calgary Corpus

5.3 Properties of the ternary sequence

The redundancy is a function increasing with the number of free parameters. For an alphabet with k symbols we have $k-1$ free parameters. Therefore we like to reduce this number as much as possible. The grouping of "close" numbers z (after MTF) improves the multi-alphabet properties of the BW-algorithm. This is important for coding of different fragments: it improves the properties of the universal coding algorithms in practice. Grouping was originally proposed (with uniform coding of all elements inside the group) for picture compression [9]. See also the theoretical study of [18]. In [8] and [2] grouping was also used, but in a different way.
Table 3 illustrates that typically the zero is the most likely symbol. To reduce the number of free parameters we like to encode as many symbols as possible with as few parameters as possible. With MTF we mix statistics from different symbols together, in other words the probability of a two can be the probability of an "a" after "t" but also the probability of a "u" after "q". More detailed it mixes also the statistic of symbols inside the same fragment. Therefore the only question arrises: "Why not a binary sequence ?"
Let as look at the situation where after a bad fragment (typically there we have a large alphabet and therefore we produce a lot of 2' th) we are starting with a run, i.e. with our rule of MTF we change

$$cdaaaaaa \qquad \rightarrow \qquad 22210000$$

or in other words only the first symbol of a run has to be encoded outside the ternary mode. In addition a bad fragment will be represented as a cluster of 2' th. Therefore in some way we are getting partial information about the switching points of the real (but still unknown) contexts.

5.4 encoding the ternary sequence

The algorithm of universal coding for a ternary Markov chain of depth $d = 3$ was chosen for describing z_1^n. Since the noisy fragment starts with the first nonzero number after a run of zeros (corresponding to a run of one symbol), we can choose the depth 1 or 2 as well, decreasing the number 3^d of states $s = z_k^{(1)}, z_{k-1}^{(1)}, z_{k-2}^{(1)}$. But the experiments show that $d = 3$ with grouping of some states together (for decreasing of redundancy) gives the best results (at least at the first step of research).
The grouping of symbols is done as follows: We transform the sequence z^n of numbers at the output of MTF into a ternary sequence z_1^n over the alphabet $\{0, 1, 2\}$. z_1^n is obtained from z by substituting all $z_k \geq 2$ by 2, i.e. we group all the numbers ≥ 2. We use these ternary sequence as a Markov chain of order 3 as a well-known universal coding scheme. This means that we implement the arithmetic coding with conditional probabilities $\vartheta(z|s(z_1^k)) = (\tau(z|s(z_1^k)) + \frac{1}{2})/(k + \frac{3}{2})$ where $z \in \{0, 1, 2\}$, $s(z_1^k) = z_k^{(1)} z_{k-1}^{(1)} z_{k-2}^{(1)}$ are the last three numbers of z_1^k, and $\tau(z|s)$ is the number of occurrences of z after the state s in z_1^k. We use this Markov chain approach to obtain information about where we are in the fragments, i.e. whether we are inside a good fragment, on the boundary of two fragments, or inside a bad fragment. The higher the order d of the Markov chain, the more information we have about the situation. But simultaneously, the statistics $\tau(z|s)$ become more and more "poor" and the redundancy of universal coding grows proportionally with the number 3^d of different states s. Therefore $d = 3$ is a reasonable choice. In addition we group some contexts together as follows:

$$\{222, 122\}, \{212, 112, 102, 012, 022, 202\}, \{121, 221\}, \{010, 110, 100\},$$

$$\{020, 200\}, \{011, 111\}$$

Each context with has not occurred forms a set with one element. We define a state for each of these sets and take statistics for each of these states independent of each other. The probabilities are then dependent only on the state and the statistic for the given state.

The use of a binary sequence let us essentially decrease the number of states. But then we must describe in z_2^n all "1" as well, and the joint rate of coding of z_1^n and z_2^n is larger then for a ternary sequence.
Now some features of such a number coding.
a) Notice that we have to take into account that we can change from one context to an other one. An improvement of the efficiency is achieved by updating the frequencies $\tau(z|s)$ and $t(a|z^k)$. All the frequencies $\tau(z|s)$ are replaced by $\left\lfloor \frac{\tau(z|s)+1}{2} \right\rfloor$ whenever one of these frequencies exceeds some fixed threshold. Updating allows to adapt (in some degree) to the change of the statistics of the encoded fragments. For a more detailed description of this updating scheme, see [2].

b) The pressure of runs is avoided by considering the frequencies of 0, 1 and 2 and the conditional probabilities for the state $s = 000$, appeared only inside the run, *separately from all the another states.*

file	*length*	*M*	*MTF0*	*MTF1*	*MTF2*
bib	111261	81	0.869	0.867	0.867
book1	768771	81	1.097	1.108	1.102
book2	610856	96	0.982	0.987	0.984
geo	102400	256	0.594	0.607	0.607
news	377109	98	1.041	1.040	1.039
obj1	21504	256	0.833	0.838	0.835
obj2	246814	256	0.939	0.934	0.933
paper1	53161	95	1.105	1.105	1.103
paper2	82199	91	1.094	1.098	1.095
pic	513216	159	0.295	0.295	0.294
progc	39611	92	1.106	1.100	1.100
progl	71646	87	0.902	0.900	0.897
progp	49379	89	0.899	0.891	0.891
trans	93695	99	0.771	0.766	0.764
	3141622		0.894	0.895	0.893

Table 5: Rates (in bits/byte) for the Calgary Corpus

c) For ternary sequence the multi-alphabet coding is not an urgent problem, especially taking into account that usually the subsequence of z_1^n, corresponding to noisy fragment, contains all three numbers.

Since the expressions for conditional probabilities were given earlier, we described the coding algorithm for z_1^n entirely.
In Table 5 we illustrate the rates for the encoding of the ternary sequence for the original MTF rule (marked as *MTF0*), for the first modified one (marked as *MTF1*) and for the second modified one (marked as *MTF2*).

5.5 Symbol coding for z_2^n

Now at last the decoder has to know which symbol has occurred at the places where we have a 2 in the ternary sequence. With the information of the ternary sequence we define a fragmentation as follows. Whenever three 0 (or more) in common have occurred (in the ternary sequence) then we say a new fragment has started. Now we can work with the statistics of the symbols instead of using the numbers generated by MTF. Remember that only the 0 at the output of MTF describe the repetitions in the original sequence. Therefore we have to use all statistics of symbols different from zero which means also the symbols represented by 1 but we have only to encode the symbols represented by a 2.

The alphabet $\mathcal{X}'$ is divided into 4 groups of variable size and content (this procedure will be described slightly later). The description of any symbol of z_2^n consists on the description of number of group, the next symbol belong to in the current moment, and to the description of symbol itself inside this group, i.e conditional probability for any symbol is the product of two conditional probabilities, as it was made for numbers.
Thus we must describe the quaternary sequence w^n of numbers of groups (of length n').

We take three statistics ($stat_1$, $stat_2$, $stat_3$) of our alphabet. all of them are updated whenever a symbol (different of 0 in the output of MTF) occurs. Now if the counter of a symbol in $stat_1$ $(stat_2, stat_3)$ achieves a threshold $l_{max,1}$ $(l_{max,2}, l_{max,3})$ then we half all entries in $stat_1$ $(stat_2, stat3$, respectively). With the halving we define a sliding window. The length of the window depends on the incoming sequence. If the alphabet size is getting smaller or one symbol is getting more likely than the threshold is achieved earlier or in other words the length of the window (memory) is getting smaller. On the other hand if we are going more to a uniform distribution on the alphabet than the memory is getting longer.

Now a symbol s is encoded if s is not on position 0 or 1 at that moment in the order of the alphabet for MTF. We use $stat_1$ if $stat_1(s) > 0$. s is encoded using $stat2$, if $stat_1(s) = 0$ and $stat_2(s) > 0$. In the same way we use $stat_3$ if $stat_1(s) = 0, stat_2(s) = 0$ and $stat_3(s) > 0$. if $stat_1(s) = stat_2(s) = stat_3(s) = 0$ then we encode the symbol on the remaining symbols uniformly.

The encoding in all these cases is done using an arithmetic coder. The only missing point is the description which statistic is used to the decoder. Therefore we define a sequence over 4 symbols. We encode a j if we use $stat_j$ (for $1 \leq j \leq 3$) and a 0 if none of the three statistics are used. We simply encode this sequence using a statistic on 4 numbers and half if a threshold is achieved. The halving is necessary because we still switch from one real context to an other one.

We define our thresholds as linear functions of the file length, i.e.

$$l_{max,1} = c_1 + \lambda_1 n, \quad l_{max,2} = c_2 + \lambda_2 l_{max,1}, \quad l_{max,3} = \lambda_3 l_{max,1} + \lambda_4 l_{max,2}.$$

5.6 Conclusion

Now we'll consider the problems, described in Section 4.6.

a) The current frequencies of symbols are defined for the subsequence of known after k steps y^k, corresponding to the subsequence of symbols of noisy fragments (containing zeros and ones but not runs). Two different approaches for updating of frequencies were discussed:

a_1). The updating with the help of threshold(s), similar to updating of frequencies for z_1^n (see above).

a_2). The another approach is decreasing generalized frequencies γ times at any step of coding. Therefore such frequency for symbol a is equal to

$$f(a|y^{k+1}) = \gamma^{-1} f(a|y^k) + \delta(a, y_{k+1}), \qquad \forall a \in A', \tag{13}$$

where $\gamma > 1$, $\delta(a, b) = 1$, if $a = b$, and $\delta(a, b) = 0$ otherwise. There are two versions of such updating.

- The updating (13) is made at any step of coding of z_2^n (or after any fixed number of steps).
- The updating (13) is made at any step of coding of y^n, but if we are inside a run then $\delta(a, b) = 0$ for any a and b.

This updating decreases the frequencies depending on the distance from the current end, where distance is calculated without runs or together with runs respectively.

b) The fact, that introduced generalized frequencies of symbols (in z_2^k) are *independent of runs*, excludes the pressure of runs entirely.

c) As it was mentioned in the previous Section, the multi-alphabet properties are introduced with the help of variable size grouping. This grouping is defined (in our case) with the help of 3 sliding windows (relative to the current end of the file) of size $L_1 < L_2 < L_3$ respectively.
The first group is the subset of different symbols in the first sliding window, the second one is the subset of different symbols in the second window except the symbols of the first one, the third group is defined similarly and the fourth group is the rest of $\mathcal{X}'$.
It is clear that the size of groups adaptively changes during coding. In particular, any group, except the first one, can be empty. Hopefully with the proper choice of L_1 the first window will define the sub-alphabet of the current uniform fragment (not exactly, of course, but at average), the aim of two an other windows is the similar one. The generalized frequencies (13) for the forth group usually are so small that they can be ignored, admitting the uniform coding of symbols of the forth group.
The coding conditional probabilities are defined in the usual way for generalized sequences and without taking into account symbols, *currently* corresponding to 0 and 1 (for the first group); 0, 1 and the elements of the first group (for the second group), etc.

This is the end of the description of mixed the coding algorithm.

In addition let us add the following. Many runs are the concatenations of runs for different contexts, but we ignore it. The fact, that just in this case the original and very simple BWT-MTF algorithm is rather efficient, means that perhaps this coding is well matched with *Non-prefix* Context Tree model of the source [16]. The noisy fragment can be the sequence of uniform fragments as well; partly it is taken into account by introducing sliding windows. But it is difficult to receive the good *estimate* of a partition of a noisy fragment into a uniform fragment for not very large files, and this is the disadvantage of the BWT.

6 Experimental Results

In an experiment we determined the compression rate of our program in bits/byte for the files of the Calgary Corpus, and compared it to other programs. Table 6 shows the compression rate of the switching method VW98 of Volf and Willems [17], of CTW (Context Tree Weighting with PPMDE, $D = 8$), of PPMDE ($D = 5$) [1], of *gzip* with option –9, of the program *BW94* developed by Burrows and Wheeler [5], of the program *F96* by Fenwick [8], of the program *BK98* developed by Balkenhol and Kurtz [2], of the program *BKS99* described in [4], and the results of the program *BS99* described in this paper.
The last row of the table shows the total length of the files and for each program the average compression rate. The two programs *VW98* and *CTW* have a better

file	*length*	*M*	*VW98*	*CTW*	*PPMDE*	*gzip*	*BW94*	*F96*	*BK98*	*BKS98*	*BS99*
bib	111261	81	1.71	1.79	1.84	2.51	2.02	1.95	1.94	1.93	1.91
book1	768771	81	2.15	2.19	2.30	3.25	2.48	2.39	2.31	2.33	2.27
book2	610856	96	1.82	1.87	1.96	2.70	2.10	2.04	2.00	2.00	1.96
geo	102400	256	4.53	4.46	4.73	5.34	4.73	4.50	4.49	4.27	4.16
news	377109	98	2.21	2.29	2.35	3.06	2.56	2.50	2.49	2.47	2.42
obj1	21504	256	3.61	3.68	3.72	3.84	3.88	3.87	3.87	3.79	3.73
obj2	246814	256	2.25	2.31	2.39	2.63	2.53	2.46	2.46	2.47	2.45
paper1	53161	95	2.15	2.25	2.31	2.79	2.52	2.46	2.45	2.44	2.41
paper2	82199	91	2.14	2.21	2.30	2.89	2.50	2.41	2.38	2.39	2.36
pic	513216	159	0.76	0.79	0.81	0.82	0.79	0.77	0.74	0.75	0.72
progc	39611	92	2.20	2.29	2.35	2.68	2.54	2.49	2.50	2.47	2.45
progl	71646	87	1.48	1.56	1.66	1.80	1.75	1.72	1.71	1.70	1.68
progp	49379	89	1.46	1.60	1.67	1.81	1.74	1.70	1.70	1.69	1.68
trans	93695	99	1.26	1.34	1.44	1.61	1.52	1.50	1.48	1.47	1.46
	3141622		2.12	2.19	2.27	2.70	2.40	2.34	2.32	2.30	2.26

Table 6: Compression rates (in bits/byte) for the Calgary Corpus

compression rate than our program: While we achieve an average compression rate of 2.260, they achieve 2.12 and 2.19, respectively. However, they and *PPMDE* require much more computational resources. If we restrict to the programs which have similar requirements in space and time (the last six columns of Table 6) then our program shows the best compression rates for all the files.

References

[1] J. Åberg. A Universal Source Coding Perspective on PPM. PhD thesis, Dept. of Information Technology, Lund Univ., 1999.

[2] B. Balkenhol and S. Kurtz. Universal Data Compression Based on the Burrows and Wheeler Transformation: Theory and Practice. Technical Report, Sonderforschungsbereich: Diskrete Strukturen in der Mathematik, Universität Bielefeld, 98-069, 1998. `http://www.mathematik.uni-bielefeld.de/sfb343/preprints/`.

[3] B. Balkenhol and S. Kurtz. Space Efficient Linear Time Computation of the Burrows and Wheeler Transformation. *I. Alth"ofer, N. Cai, G. Dueck, L. Khachatrian, M. Pinsker, A. Sark"ozy, I. Wegener and Z. Zhang (eds.), Numbers, Information and complexity (Festschrift in honour of Rudolf Ahlswede's 60th Birthday)*, pages 375–384, 1999.

[4] B. Balkenhol, S. Kurtz, and Y.M. Shtarkov. Modifications of the Burrows and Wheeler Data Compression Algorithm. In *Proceedings Data Compression Conference, Snowbird, Utah, March 29-31*, pages 188–197, 1999. `http://www.mathematik.uni-bielefeld.de/~bernhard/`.

[5] M. Burrows and D.J. Wheeler. A Block-Sorting Lossless Data Compression Algorithm. Research Report 124, Digital Systems Research Center, 1994. `http://gatekeeper.dec.com/pub/DEC/SRC/research-reports/abstracts/src-rr-124.html`.

[6] B. Chapin and S.R. Tate. Higher Compression from the Burows Wheeler Transform by Modified Sorting. Technical Report, University of North Texas, Department of Computer Science, 1998. `http://www.cs.unt.edu/~srt/papers/bwtsort.pdf`.

[7] J.G. Cleary, W. Teahan, and I.H. Witten. Unbounded Length Contexts for PPM. In *Proceedings of the IEEE Data Compression Conference, Snowbird, Utah*, pages 52–61. IEEE Computer Society Press, 1995.

[8] P. Fenwick. Block Sorting Text Compression. In *Proceedings of the 19th Australian Computer Science Conf., Melbourne, Australia, Jan. 31 - Feb. 2, 1996*, 1996.

[9] O. Franceschi, Y.M. Shtarkov, and R. Forchheimer. An Adaptive Coding Method for Still Images. In *Picture Coding Symp, PCS-88, Torini, Italy*, pages 6.5–1 – 6.5–2, 1988.

[10] N.J. Larsson. The Context Trees of Block Sorting Compression. In *Proceedings of the IEEE Data Compression Conference, Snowbird, Utah, March 30 - April 1*, pages 189–198. IEEE Computer Society Press, 1998.

[11] N.J. Larsson. Structures of String Matching and Data Compression. PhD thesis, Dept. of Comp. Science, Lund Univ., 1999.

[12] N.J. Larsson and K. Sadakane. FastSuffix Sorting. Technical Report LU-CS-TR:99-214, LUNDFD6/(NFCS-3140)/1-20/(1999), Dept. of Comp. Science, Lund Univ., 1999. `http://www.dna.lth.se/~jesper/tr214.ps.gz`.

[13] K. Sadakane. A Fast Algorithm for Making Suffix Arrays and for Burrows-Wheeler Transformation. In *Proceedings of the IEEE Data Compression Conference, Snowbird, Utah, March 30 - April 1*, pages 129–138. IEEE Computer Society Press, 1998.

[14] M. Schindler. A Fast Block-Sorting Algorithm for Lossless Data Compression. Technical report, 1996. `http://www.compressconsult.com/szip/`.

[15] Y.M. Shtarkov. Switching Discrete Sources and its Universal Coding. *PPI*, **28**(3):95–111, 1992.

[16] Y.M. Shtarkov. Universal Coding of Non-Prefix Context Tree Sources. *I. Alth"ofer, N. Cai, G. Dueck, L. Khachatrian, M. Pinsker, A. Sark"ozy, I. Wegener and Z. Zhang (eds.), Numbers, Information and complexity (Festschrift in honour of Rudolf Ahlswede's 60th Birthday)*, pages ???–???, 1999.

[17] P.A.J. Volf and F.M.J. Willems. The Switching Method: Elaborations. In *Proc. 19-th Symp. Inform. Theory in the Benelux, Veldhoven, The Netherlands, May 28-29*, pages 13–20, 1998.

[18] F.M.J. Willems. Universal Data Compression and Repetition Times. *IEEE Trans. on Inform. Theory*, **35**(1):54–58, 1989.