Universal Lossless Source Coding with the Burrows Wheeler Transform*

Michelle Effros

California Institute of Technology

Abstract

We here consider a theoretical evaluation of data compression algorithms based on the Burrows Wheeler Transform (BWT). The main contributions include a variety of very simple new techniques for BWT-based universal lossless source coding on finite-memory sources and a set of new rate of convergence results for BWT-based source codes. The result is a theoretical validation and quantification of the earlier experimental observation that BWT-based lossless source codes give performance better than that of Ziv-Lempel style codes and almost as good as that of prediction by partial mapping (PPM) algorithms.

I Introduction

The Burrows Wheeler Transform (BWT) [1] is a slightly expansive reversible sequence transformation currently receiving considerable attention from researchers interested in practical lossless data compression. To date, the majority of the research devoted to BWT-based compression algorithms has focused on experimental comparisons of BWT-based algorithms with competing codes. Experimental results on algorithms using this transformation (e.g., [2]) indicate lossless coding rates better than those achieved by Ziv-Lempel codes (LZ'77 and LZ'78) but typically not quite as good as those achieved by the prediction by partial mapping (PPM) schemes described in works such as [3]. Further, BWT code implementation yields codes with complexity comparable to that of the Ziv-Lempel codes, which are significantly faster than algorithms like PPM. To date, these comparisons have primarily involved experimental studies of implementation speeds and achieved lossless description rates.

This work presents an information theoretic look at lossless source codes based on the BWT. The key results of this analysis are: a rate of convergence result for an existing universal BWT-based code for finite-memory sources, introduction of a variety of new universal BWT-type codes, an analysis of the rate of convergence bounds for the coding performance of each new algorithm, and a comparison of BWT-based codes to each other and to other universal coding algorithms. The result of this comparison is a confirmation and quantification of the experimentally observed

*This material is based upon work partially supported by NSF CAREER Grant No. MIP-9501977, a grant from the Powell Foundation, and donations through the Intel 2000 Technology for Education Program.

results. On sequences of length n, the performance of the best BWT-based codes converges to the optimal performance at a rate of $O(\log n/n)$ (within a constant factor of the optimal performance) and the simplest universal BWT codes converge at rates of $O(\sqrt{\log n/n})$. These rates of convergence are better than the $O(\log\log n/\log n)$ convergence [4] of LZ'77 or the $O(1/\log n)$ convergence [5, 6] of LZ'78.

The remainder of this work is organized as follows. Section II contains a variety of background material relevant to this work. Included in that section are: an introduction to universal source coding, a description of the class of finite-memory sources (called FSMX sources) treated in this work, and a brief description of previous universal coding results for FSMX sources. Section III contains a description of the BWT. In Section IV, we consider the statistical properties of BWT-transformed data sequences from FSMX sources. In Section V, we describe a variety of extremely simple new BWT-based lossless source codes, prove their universality, and derive rate of convergence results. A theoretical analysis of an earlier approach to coding BWT-transformed data sequences is also included. Complete details on the proofs of Section V appear in [7]. The rate of convergence results for the algorithms introduced here range from $O(\sqrt{\log n/n})$ for the codes requiring the least memory and computation to $O(\log n/n)$ for a more complex BWT-based algorithm or a BWT-based code with access to information about the length of the source's finite memory. Thus even the simplest BWT-based code gives a rate of convergence faster than either of the Ziv-Lempel algorithms while the BWT code with the fastest rate of convergence achieves – to within a constant factor – the optimal rate of convergence. A summary of results and conclusions appears in Section VI.

II Background and Definitions

A universal lossless source code is a sequence of source codes that asymptotically achieves the optimal performance for every source in some broad class of possible sources. Making this notion more precise requires some definitions.

Consider any class $\{P_\theta : \theta \in \Lambda\}$ of stationary ergodic sources on finite source alphabet $\mathcal{X}$. For each $\theta \in \Lambda$, let $H_\theta(X^n)$ and $H_\theta(\mathcal{X})$ be the nth-order entropy and the entropy rate respectively of P_θ. Thus $H_\theta(X^n) = \sum_{x^n \in \mathcal{X}^n}[-P_\theta(x^n)\log P_\theta(x^n)]$ and $H_\theta(\mathcal{X}) = \lim_{n\to\infty}(1/n)H_\theta(X^n)$ for each $\theta \in \Lambda$. Given any variable-rate lossless source coding strategy for coding n-sequences from $\mathcal{X}$, for each $x^n = (x_1, \ldots, x_n) \in \mathcal{X}^n$, let $\ell(x^n)$ be the description length achieved in the lossless description of x^n. For each $\theta \in \Lambda$, we use $\delta_n(\theta)$ to represent the expected redundancy associated with using the given code to code samples from distribution P_θ. That is, $\delta_n(\theta) = E_\theta \ell(X^n)/n - H_\theta(X^n)/n$ is the difference between the expected rate per symbol using the given blocklength-n code and the optimal rate per symbol for coding n-vectors from P_θ. We call a sequence of coding strategies a *weakly minimax universal lossless source code* on Λ if $\delta_n(\theta) \to 0$ for each $\theta \in \Lambda$. In [8], Rissanen demonstrates that for any class Λ of sources parameterized by K real numbers, the optimal rate of convergence of $(K/2)\log n/n$ is achievable to within $O(1/n)$ for almost all $\theta \in \Lambda$.

We here consider the problem of minimax universal lossless source coding for finite-memory sources. In particular, we consider a subset of the class of unifilar, ergodic, finite-state-machine (FSM) sources. An FSM source is defined by a finite alphabet $\mathcal{X}$, a finite set of states $\mathcal{S}$, $|\mathcal{S}|$ conditional probability measures $p(x|s)$, $x \in \mathcal{X}$, $s \in \mathcal{S}$, and a next-state function f mapping $\mathcal{S} \times \mathcal{X}$ into $\mathcal{S}$. Given an FSM data source and an initial state s_0, the probability of string $x^n = x_1, \ldots, x_n \in \mathcal{X}^n$ is defined as $P(x^n) = \prod_{i=1}^{n} p(x_i|s_{i-1})$, where $s_i = f(s_{i-1}, x_i)$ for all $1 \leq i \leq n$.

The class of FSMX sources is the subset of the class of FSM sources for which there exists an integer L such that for every $n \geq L$, the state s_n is uniquely determined by $x_{n-L+1}, x_{n-L+2}, \ldots, x_n$. For FSMX sources, the set $\mathcal{S}$ is defined by a minimum suffix set of strings from $\mathcal{X}$ with the property that for every string $s \in \mathcal{S}$ and every $x \in \mathcal{X}$ such that $p(x|s) \neq 0$, the string sx has exactly one suffix in S. We use the notation $\mathrm{suf}(sx)$ to denote the suffix of the string formed by concatenating $s \in \mathcal{S}$ and $x \in \mathcal{X}$. Using this notation, the next-state function $f(s, x)$ becomes $f(s, x) = \mathrm{suf}(sx)$ for all $s \in \mathcal{S}$ and $x \in \mathcal{X}$. Examples of FSMX sources, also called finite-order FSM sources, include Markov-k sources for any $k \geq 0$.

In [9], Rissanen proves the existence of universal source codes for *binary* FSM sources with an unknown number $|\mathcal{S}|$ of states, achieving $\delta_n(\theta) = (|\mathcal{S}|/2)\log n/n + O(1/n)$ for almost all θ. The optimal algorithm traverses the entire data sequence to determine its best estimate of $\mathcal{S}$ and then describes the data sequence using that estimate. An alternative to Rissanen's algorithm with the same rate of convergence on FSMX sources with *known* memory constraint L appears in [10]. The algorithm computes and sequentially updates an on-line estimate of $\mathcal{S}$ during the coding process.

III The Burrows Wheeler Transform

The Burrows Wheeler Transform [1] is a reversible block-sorting transform that operates on a sequence of n data symbols to produce a permuted data sequence of the same symbols and a single integer in $\{1, \ldots, n\}$. Let $BWT_n : \mathcal{X}^n \rightarrow \mathcal{X}^n \times \{1, \ldots, n\}$ denote the n-dimensional BWT function and $BWT_n^{(-1)} : \mathcal{X}^n \times \{1, \ldots, n\} \rightarrow \mathcal{X}^n$ denote the inverse of BWT_n. We write $(y^n, m) = BWT(x^n)$ and $BWT^{(-1)}(y^n, m) = x^n$. The notations $BWT_{\mathcal{X}}$ and $BWT_{\mathbb{N}}$ denote the character and integer portions, respectively.

The forward BWT proceeds by forming all n cyclic shifts of the data string and sorting those cyclic shifts lexicographically. The BWT output is the last character of each of the (ordered) cyclic shifts followed by an integer describing the location of the original data sequence in the ordered list. An example appears in Figure 1.[1]

Intuitively, the inverse BWT starts with the last column of the table of lexicographically ordered cyclic shifts and builds up the rest of the table. By the table construction, the first column of the table is simply an ordered copy of the last column of the table. Thus given the table's last column – that is the transform output – we can immediately reconstruct the first column by alphabetizing the same sequence.

[1]A variation of the BWT appends a unique "end-of-file" (EOF) symbol to the end of the data sequence x^n. Inclusion of the EOF symbol removes the need to describe the row index n and buffers the end of the string from its beginning, but simultaneously increases the alphabet size by one. We here concentrate on the variation described in the text above.

Input: banana Output: (nnbaaa,4)

	Step 1						Step 2						Step 3
1	**b**	**a**	**n**	**a**	**n**	**a**	a	b	a	n	a	n	n
2	a	n	a	n	a	b	a	n	a	b	a	n	n
3	n	a	n	a	b	a	a	n	a	n	a	b	b
4	a	n	a	b	a	n	**b**	**a**	**n**	**a**	**n**	**a**	a
5	n	a	b	a	n	a	n	a	b	a	n	a	a
6	a	b	a	n	a	n	n	a	n	a	b	a	a

Figure 1: The BWT of the sequence "banana". The original data sequence appears in row 4 of the lexicographically ordered table (Step 2); the final column of that table contains the sequence "nnbaaa". Hence $BWT(\text{banana}) = (\text{nnbaaa}, 4)$.

Further, since each row is a cyclic shift of every other row, the last and first columns together provide a list of all consecutive pairs of letters. An ordered list of these pairs gives the first and second table columns. Repeating the above process for all triples, quadruples, etc. sequentially reproduces all columns of the original data matrix. Finally, the transform index indicates the appropriate row of the completed table. An example of the inverse BWT appears in Figure 2.

A memory- and complexity-efficient implementation of the BWT is used in data compression algorithms in [1] and later work derived from it. In each algorithm, the data is first transformed using the BWT and then compressed using a lossless source code. Typically, these algorithms capitalize on the BWT's tendency to group together long strings of like characters, thereby producing a string that is more easily compressed than the original data sequence. Yet of all of the columns in the table, the last column has the least effect on the lexicographic ordering of the rows and is thus, in some sense, the *least* ordered of the columns. It is therefore tempting to suggest an alternative to the BWT that simply uses some other matrix column as the transform output. Unfortunately, for general strings and sequence lengths, the last column is the only column that yields a reversible transformation.

IV The BWT on FSMX Sources

Consider the properties of a string that has been time-reversed and then transformed with the BWT. The resulting data sequence has the property that characters that *follow* like strings – e.g., characters with a common prefix – are grouped together. Since FSMX sources have the property that characters with the same prefix are drawn from the same distribution, this reordering is extremely useful for data compression.

For any source sequence $X^n = X_1, X_2, \ldots, X_n$ from alphabet $\mathcal{X}$, let $Y^n = \mathcal{R}(X^n)$ and $(Z^n, M) = BWT(Y^n) = BWT(\mathcal{R}(X^n))$, where $\mathcal{R}_n : \mathcal{X}^n \to \mathcal{X}^n$ is the time-reversal operator. Thus $Y^n = (Y_1, \ldots, Y_n) = (X_n, \ldots, X_1)$, and Z^n and M are the BWT data sequence and the row index respectively of the reversed data string.

The following definitions are useful for understanding the statistical properties

Input: (nnbaaa,4) Output: banana

1	2	3	4	5	6	7	8	9	10
n	a···n	na	ab···n	nab	aba···n	naba	aban···n	naban	abanan
n	a···n	na	an···n	nan	ana···n	nana	anab···n	nanab	anaban
b	a···b	ba	an···b	ban	ana···b	bana	anan···b	banan	ananab
a	**b···a**	ab	**ba···a**	aba	**ban···a**	aban	**bana···a**	abana	**banana**
a	n···a	an	na···a	ana	nab···a	anab	naba···a	anaba	nabana
a	n···a	an	na···a	ana	nan···a	anan	nana···a	anana	nanaba

Figure 2: The inverse BWT for the pair (nnbaaa,4). The output of the inverse BWT is row $BWT_{\mathbb{N}}$=4 of the final table. The inverse BWT may be sequentially decoded.

of Z^n when X^n is drawn from an FSMX source. Given a random vector $U^n = (U_1, U_2, \ldots, U_n)$ and any integer $1 \leq C \leq n$, we call U^n *C-piecewise independent and identically distributed* (C-p.i.i.d.) if there exists some collection $\{p_1, p_2, \ldots, p_C\}$ of distributions on $\mathcal{X}$ such that for any $x^n \in \mathcal{X}^n$ there exists an integer transition pattern $T(x^n) = (T_1, T_2, \ldots, T_{C+1})$, $1 = T_1 < T_2 < T_3 \leq \cdots < T_{C+1} = n+1$, such that $\Pr(U^n = x^n) = \prod_{j=1}^{C} \prod_{i=T_j}^{T_{j+1}-1} p_j(x_i)$. Thus U^n C-p.i.i.d. implies that U^n looks like a concatenation of C iid strings. Notice, however, that the elements of U^n need not actually be independent since, for example, the transition indices need not be independent. We therefore make a distinction between sources that are p.i.i.d. and sources that are independent and piecewise identically distributed (i.p.i.d.), where a string is C-i.p.i.d. if it is C-p.i.i.d. and its elements are independent.

Lemma 1 *For any sequence length n and any X^n drawn according to an arbitrary FSMX distribution with minimum suffix set $\mathcal{S}$ and initial state s_0, the string $Z^n = BWT_{\mathcal{X}}(\mathcal{R}(X^n))$ is C-p.i.i.d. with $C \leq |\mathcal{S}|$.*

Thus the BWT of the time-reversed data string looks like a list of iid samples with a number of parameter changes less than the number of states in the FSMX source. This property is achieved by the BWT on *any* FSMX source independent of the suffix set $\mathcal{S}$ and plays a crucial role in the success of BWT-based source codes.

V Universal Coding on Finite Memory Sources

All but one of the combinations of the BWT and a lossless source code considered in this section are new, as are all of the rate of convergence results. The remaining algorithm, treated first, is a variation on the BWT-based lossless source code in common use for practical coding. Each code describes the integer and ordered string parts of a BWT using independent lossless source codes. We reverse all data strings prior to transformation by the BWT to group characters with common prefixes together.

Let $X_1, X_2, \ldots$ be drawn according to an FSMX source. Each of the algorithms considered uses $\lceil \log n \rceil$ bits to describe the random variable $BWT_{\mathbb{N}}(\mathcal{R}(X^n))$. The algorithms differ in their methods for describing $BWT_{\mathcal{X}}(\mathcal{R}(X^n))$. Since all of the algorithms code Z^n rather than coding X^n directly, none of the algorithms is a sequential code. That is, each algorithm requires access to the complete vector X^n before coding even the first symbol X_1. Nonetheless, the BWT is extremely computationally efficient, and most of the algorithms considered use very low complexity sequential coding of the BWT output Z^n. The algorithms are proven to be universal for FSMX sources and most do not require a priori knowledge of the memory constraint L (as in [10]) or state space $\mathcal{S}$. We here focus on $\Delta_n(\theta) = E_\theta \ell(X^n)/n - H_\theta(\mathcal{X})$ rather than $\delta_n(\theta)$. Note that $\Delta_n(\theta) = \delta_n(\theta) + O(1/n)$ for FSMX sources.

Most of the algorithms make use of arithmetic codes with the Krichevsky and Trofimov [11] probability model. The resulting codes are extremely efficient in memory and computation, requiring only $|\mathcal{X}|$ counters. The description length on an arbitrary string x^n is bounded as $\ell(x^n) \leq nH(\hat{\theta}(x^n)) + \frac{|\mathcal{X}|-1}{2}\log n + c$, where $H(\hat{\theta}(X^n))$ is the entropy associated with a memoryless source $P_{\hat{\theta}(X^n)}$ matching the single-letter empirical distribution of x^n.

A Move-to-Front Code – The Baseline Source Code

We begin by analyzing a variation on the practical BWT-based codes described in works like [2, 12, 13]. The algorithm uses an integer code requiring no more than $\log^* j + c$ bits to describe integer j [14], where $\log^* j = (\log j)^+ + (\log\log j)^+ + \ldots$ (x^+ gives x if x is defined and nonnegative and 0 otherwise) and $\log^* j + c \leq (\log j)^+ + (\log\log j)^+ + 2(\log\log\log j)^+ + 7$. Following the argument of Elias [15], initialize a memory system with some ordered list containing one copy of each letter in $\mathcal{X}$ and then code each symbol Z_k by describing the interval since the last appearance of Z_k. While this algorithm is not universal when performed on alphabet $\mathcal{X}$, it can be universal if performed on extensions $\mathcal{X}^m$ of $\mathcal{X}$ with $m \to \infty$, as discussed in [16] for Markov-k sources.[2] The proof used here takes a different approach from the typicality arguments of [16] in order to derive rate of convergence results.

Theorem 1 *Given an FSMX source with unknown state space $\mathcal{S}$ and memory constraint $L < \infty$, the Elias interval code of the BWT of the reversed data string achieves redundancy $\Delta_n(\theta) \leq \log^* H(\mathcal{X}) + c + O(\log n/n)$ bits per symbol for all θ in the given class. Applying the same code to alphabet $\mathcal{X}^m$, $m \to \infty$ gives a universal code with redundancy bounded as*

$$\Delta_n(\theta) \leq \frac{\log\log n}{\log n} + O\left(\left(\frac{\log\log n}{\log n}\right)^2\right)$$

bits per symbol for all θ in the given class.

While the baseline code redundancy does not converge to zero, the code is very simple, and for practical n-values the constant to which the redundancy converges seems to be quite benign. In contrast, the extension code yields a redundancy approaching

[2]The result of [16] is correct, but the argument is misleading. The alphabet size is discarded as finite in Theorem 2 ($m = 1$), which is then applied to alphabet $\mathcal{X}^m$ with $m \to \infty$ to prove universality in Theorem 3.

zero, but the code is memory- and complexity-inefficient. Theorems 2 and 3 use no alphabet extensions and *extremely* simple and memory-efficient source codes.

Known State Space $\mathcal{S}$ or Memory Constraint L

Consider an FSMX source $X_1, X_2, \ldots$ with *known* state space $\mathcal{S}$. By Lemma 1, $BWT_{\mathcal{X}}(\mathcal{R}(X^n))$ is $|\mathcal{S}|$-p.i.i.d., and for each $j \in \{1, \ldots, |\mathcal{S}|-1\}$, the boundary T_{j+1} between distribution p_j and distribution p_{j+1} is immediately apparent at the encoder – which has access to all of the information contained in the BWT table. As a result, universal coding performance can be achieved by explicitly describing these boundary points to the decoder and then independently encoding the subsequences divided by these boundary points with an arbitrary universal source code. For any FSMX source with $|\mathcal{S}|$ states, we describe $|\mathcal{S}|-1$ transition points, each of value between 1 and n. Thus the rate associated with describing the boundary points to the decoder is no greater than $(|\mathcal{S}|-1)\lceil\log n\rceil$.[3] The total description length of the scheme equals $|\mathcal{S}|\lceil\log n\rceil$ bits to describe $BWT_{\mathbb{N}}(\mathcal{R}(X^n))$ and the $|\mathcal{S}|-1$ transition points plus the rate used in describing the data given the transition points. We apply the Krichevsky-Trofimov code to describe the data given the transition points.

Theorem 2 *Given an FSMX source with known state space $\mathcal{S}$, the arithmetic code that independently uses the Krichevsky-Trofimov distribution on each substring of the BWT of the reversed data string is strongly minimax universal with redundancy*

$$\Delta_n(\theta) \leq \frac{|\mathcal{S}|(|\mathcal{X}|+1)}{2}\frac{\log n}{n} + O\left(\frac{1}{n}\right)$$

bits per symbol for all θ in the given class. When the state space $\mathcal{S}$ is unknown, but L is known, coding as if $|\mathcal{S}|$ were equal to $|\mathcal{X}|^L$ gives $\Delta_n(\theta) \leq (|\mathcal{X}|^L(|\mathcal{X}|+1)/2)\log n/n + O(1/n)$ bits per symbol for all θ in the given class.

The rate of convergence described in Theorem 2 differs from Rissanen's optimal rate of convergence by a constant factor of $(|\mathcal{X}|+1)/(|\mathcal{X}|-1)$. For very small $|\mathcal{X}|$ (e.g., a binary source), this factor grows as large as 3. (We restrict our attention to nontrivial examples, requiring $|\mathcal{X}| \geq 2$.) This factor shrinks to 1 for large alphabets, and thus can be made arbitrarily small (at the cost of a higher complexity) by coding vector extensions of the original alphabet. Most of the redundancy increase comes from the boundary point descriptions while the rest is the cost of describing the appropriate row in the BWT decoding table. This extra redundancy likely results from the fact that while our string is p.i.i.d., the symbols are not independent. In particular, the parameters are jointly distributed, and this fact is not exploited either here or in any of the algorithms considered in this work. Unfortunately algorithms that take this redundancy into account seem to be quite complex. We therefore favor the above approach, which, while non-optimal in its rate of convergence lies within a constant factor of the optimum and is achieved without the use of large dictionaries of probability coefficients like those found in most universal codes.

[3] We can actually do better. More sophisticated (but also more complex) codes would exploit the relationships between these boundary points, which are not independent. We here stick to the simplest approach.

When the memory constraint L is known but the minimal state space $\mathcal{S}$ is unknown, the above strategy gives a rate of convergence that differs from the optimal by at most a factor of $|\mathcal{X}|^L(|\mathcal{X}|+1)/(|\mathcal{S}|(|\mathcal{X}-1|))$. Thus the algorithm has the disadvantage of a larger constant in front of the redundancy bound. Note, however, that the code requires no more memory or computation when $\mathcal{S}$ is unknown than when $\mathcal{S}$ is known since the algorithm requires the tracking of only one distribution at a time. This property is not shared by the (PPM-style) code that takes the same approach in the original sequence domain rather than the BWT domain.

Unknown State Space $\mathcal{S}$ and Memory Constraint L

While the above code is conceptually simple and achieves a good rate of convergence, it is impractical in its assumption of knowledge of either the FSMX states or the memory constraint L. We focus on several extremely simple alternatives.

A Finite Memory Code

While it is possible for the encoder to attempt to estimate $\mathcal{S}$ and then describe the resulting $T(Z^n)$, such codes are often quite complex. Notice, however, that the number of transition points is finite and independent of the sequence length n. We therefore encode the data using the Krichevsky-Trofimov coder but keeping the memory of the code finite. That is, for a sequence of length n and some maximum memory length $w(n)$, we use the Krichevsky-Trofimov code $\lceil n/w(n) \rceil$ times: Once on symbols $Z_1^{w(n)} = Z_1, \ldots, Z_{w(n)}$, next on symbols $Z_{w(n)+1}^{2w(n)} = Z_{w(n)+1}, \ldots, Z_{2w(n)}$, and so on until the entire data sequence is coded. The "window" length $w(n)$ must be carefully chosen to grow as a function of n (so that the per-symbol redundancy on each length $w(n)$ sequence goes to zero) but not too quickly (so that the fraction of windows in which we find two or more distributions in the p.i.i.d. string is small). Theorem 3 bounds the redundancy for the optimal window length $w(n)$.

Theorem 3 *Given an FSMX source with unknown state space $\mathcal{S}$ and memory constraint $L < \infty$, the arithmetic code that uses, on each substring of the BWT of the reversed data string, the Krichevsky-Trofimov distribution with a fixed-length finite memory is strongly minimax universal with redundancy*

$$\Delta_n(\theta) \leq \sqrt{(|\mathcal{S}|-1)(|\mathcal{X}|-1)\log|\mathcal{X}|}\sqrt{\frac{\log n}{n}} + O\left(\frac{\log\log n}{\sqrt{n\log n}}\right)$$

bits per symbol for all θ in the given class if the choice of $w(n)$ is allowed to depend on the $|\mathcal{S}|$. For the more general problem, where $|\mathcal{S}|$ is unknown and therefore cannot be used in the choice of a window length, $\Delta_n(\theta) \leq O\left(\sqrt{\log n/n}\right)$.

The FSMX source model and the resulting analysis do not take into account several fortuitous characteristics of the above described algorithm. In particular, for many sources, such as text, neighboring distributions in the BWT output will often have very similar statistics due to their similar prefixes. Thus even in regions overlapping more than one distribution, the performance of this code is expected to be quite good.

An alternative to the above finite-memory approach would be a sliding window approach. The sliding window code uses the same arithmetic coding structure but always bases its probability estimate on the past $w(n)$ samples.

More Sophisticated Source Codes

While this work focuses primarily on *extremely simple* universal source coding with the BWT, we here briefly introduce a variety of other, more complex approaches to coding BWT-transformed data sets and give bounds on their performances.

In [17], Merhav introduces a universal lossless code for sources with piecewise-constant parameters. Applying Merhav's code to BWT-transformed data sequences yields a total redundancy of $(|\mathcal{S}|+1)\log n/n+|\mathcal{S}|(|\mathcal{X}|-1)\log n/(2n)+O(1/n)$. Unfortunately, when $|\mathcal{S}|$ is unknown, Merhav's algorithm can become extremely complex since it involves weighting over all possible transition patterns and the number of transitions patterns to be considered grows exponentially in the sequence length [18].

In [18], Willems describes two techniques for coding binary i.p.i.d. data sequences. The first algorithm, a quadratic complexity code, achieves the same rate of convergence as Merhav's code. The second, a linear complexity code, achieves a rate of convergence at most a factor of 1.5 higher. Both complexity bounds are independent of $|\mathcal{S}|$. Using these algorithms in our BWT source codes results in universal codes with rates of convergence $(|\mathcal{S}|+1)\log n/n + |\mathcal{S}|(|\mathcal{X}|-1)\log n/(2n) + O(1/n)$ and $(3|\mathcal{S}|/2+1)\log n/n + |\mathcal{S}|(|\mathcal{X}|-1)\log n/(2n) + O(1/n)$, respectively.

VI Conclusions

The BWT lends itself to a wide variety of universal lossless source codes for FSMX sources. As the BWT requires processing of an entire data sequence, BWT-based codes are not sequential, but often employ sequential coding of the transformed data sequences. The resulting codes yield good theoretical and practical coding performance. The specific algorithms introduced here yield rates of convergence between those of Ziv-Lempel codes and those of PPM codes, potentially with very low complexity. Further, even on FSMX sources for which the number $|\mathcal{S}|$ of states is large, the codes discussed do not require the enormous dictionaries of conditional probabilities associated with many algorithms yielding comparable performance.

Acknowledgments Thanks go to Dmitre Linde and Jeremy Kahn for their input.

References

[1] M. Burrows and D. J. Wheeler. A block-sorting lossless data compression algorithm. Technical Report SRC 124, Digital Systems Research Center, Palo Alto, CA, May 1994.

[2] J. G. Cleary, W. J. Teahan, and I. H. Witten. Unbounded length contexts for PPM. In *Proceedings of the Data Compression Conference*, pages 52–61, Snowbird, UT, March 1995. IEEE Computer Society.

[3] J. G. Cleary and I. H. Witten. Data compression using adaptive coding and partial string matching. *IEEE Transactions on Communications*, 32(4):396–402, April 1984.

[4] A. D. Wyner and A. J. Wyner. Improved redundancy of a version of the Lempel-Ziv algorithm. *IEEE Transactions on Information Theory*, IT-41(3):723–732, May 1995.

[5] G. Louchard and W. Szpankowski. On the average redundancy rate of the Lempel-Ziv code. *IEEE Transactions on Information Theory*, IT-43(1):1–8, January 1997.

[6] S. A. Savari. Redundancy of the Lempel-Ziv incremental parsing rule. *IEEE Transactions on Information Theory*, IT-43(1):9–21, January 1997.

[7] M. Effros. Universal Burrows-Wheeler source coding. 1998. In preparation.

[8] J. Rissanen. Universal coding, information, prediction, and estimation. *IEEE Transactions on Information Theory*, 30(4):629–636, July 1984.

[9] J. Rissanen. Complexity of strings in the class of Markov processes. *IEEE Transactions on Information Theory*, IT-32(4):526–532, July 1986.

[10] M. Weinberger, A. Lempel, and J. Ziv. A sequential algorithm for the universal coding of finite memory sources. *IEEE Transactions on Information Theory*, IT-38(3):1002–1014, May 1992.

[11] R.E. Krichevsky and V.K. Trofimov. The performance of universal encoding. *IEEE Transactions on Information Theory*, IT-27(2):199–207, 1981.

[12] Z. Arnavut and S. S. Magliveras. Lexical permutation sorting algorithm. *The Computer Journal*, 40(5):292–295, 1997.

[13] N. J. Larsson. The context trees of block sorting compression. In *Proceedings of the Data Compression Conference*, pages 189–198, Snowbird, UT, March 1998. IEEE.

[14] T. M. Cover and J. A. Thomas. *Elements of Information Theory.* Wiley, 1991.

[15] P. Elias. Interval and recency rank source coding: two on line adaptive variable-length schemes. *IEEE Transactions on Information Theory*, IT-33(1):3–10, January 1987.

[16] M. Arimura and H. Yamamoto. Asymptotic optimality of the block sorting data compression algorithm. *IEICE Trans. Fundamentals*, E81-A(10):2117–2122, October 1998.

[17] N. Merhav. On the minimum description length principle for sources with piecewise constant parameters. *IEEE Transactions on Information Theory*, IT-39(6):1962–1967, November 1993.

[18] F. M. J. Willems. Coding for a binary independent piecewise-identically-distributed source. *IEEE Transactions on Information Theory*, IT-42(6):2210–2217, November 1996.