Codes for Unordered Sets of Words

Yuriy A. Reznik
Qualcomm Inc., San Diego, CA
Email: yreznik@ieee.org

***Abstract*—We study the problem of coding of unordered sets of words, appearing in language processing, retrieval, machine learning, computer vision, and other fields. We review known results about this problem, and offer a code construction technique suitable for solving it. We show that in a memoryless model the expected length of our codes approaches $Ht - \log m! + O(m)$ where m is the number of words in the set, t is the combined length of all words, and H is the entropy of the source. We also offer design of a universal code for sets of words and perform its redundancy analysis.**

I. Introduction

The classic problem of source coding is to encode a given *sequence of words* (or a *message*) $w = w_1, w_2, \ldots$ such that the length of the code is minimal. Most commonly, it is further assumed that words must be decoded in the same order as they appear, and that the result of decoding must be unique and matching the original. This setting, coupled with the assumption about stochastic nature of the source, has lead to many fundamental results, including Shannon's source coding theorem, Huffman codes, and others [1].

Nevertheless, in practice we may also encounter a slightly different problem: the message may be given by a *set of words* $\{w_1, \ldots, w_m\}$, where their order is not important. This happens, for example, when we formulate a request to a search engine by providing a list of keywords. Such keywords can be communicated in any order, without affecting the meaning of the message. Given this flexibility, one may expect a code constructed for a set $\{w_1, \ldots, w_m\}$ to consume about $\log_2 m!$ less bits than a code constructed for a particular sequence $w_{i_1}, \ldots, w_{i_m}$. However, the construction of codes for unordered sets does not appear to be entirely trivial: most existing source coding tools assume sequential processing. Somewhat different approach is needed in this case.

The purpose of this paper is to offer one possible method for solving this coding problem. The key tool that we employ comes from information retrieval: it is a *digital search tree* or *DST* structure, due to E. Coffman and J. Eve [9]. It is similar, in a sense, to the *prefix tree* or *incremental parsing rule* of J. Ziv and A. Lempel's code [19]. However, the prefix tree always parses a single sequence, while the DST is designed to parse a *set* of sequences. DST is also different than parsing trees used by Tunstall or Khodak codes [20]–[22], CTW [23], and other conventional source coding algorithms. Once the DST is constructed, we use Zacks ranking scheme [24] to compute its lexicographic index, and then we transmit it. Finally, to encode parts of input words that are not "absorbed" by the tree structure, we define a canonic order of nodes in the tree, and transmit missing parts of words according to this order. We provide detailed analysis of the average performance of this scheme in the memoryless model, and show that, under certain conditions, it asymptotically approaches the expected $\log_2 m!$ reduction in the bitrate. We also describe and analyze design of universal codes based on DST-encoding technique.

Among related prior studies, we must mention 1986 work of A. Lempel [5], who predicted existence of *multiset decipherable codes*, and shown that such codes should be more compact than conventional (uniquely decipherable) codes when $m > 3$. Another related class of codes (for words over partially commutative alphabet) was studied by S. Savari [6]. The achievable performance bounds for coding of sets were studied by L. Varshney and V. Goyal [7]. Properties of DST and related structures were studied in [3], [10]–[16]. Techniques for coding of trees were discussed in [24]–[26]. Descriptions of other known uses of coding of trees in data compression can be found in [29]–[31]. The problem of construction of codes for unordered sets was first considered by the author in [8]. This paper is an extension of that work.

In Section II we provide detailed description of our technique. In Section III, we study asymptotic performance of the resulting coding scheme in the memoryless model. Section IV shows how our scheme can be adopted to construct universal codes. Conclusions are drawn in Section V.

II. Coding of Sets of Words

Let $\{w_1, \ldots, w_m\}$ be a set of words that we need to encode. For simplicity, we will assume that these words are binary, of length $|w_i| = n$, and produced by a *symmetric memoryless source*. That is, characters "0" and "1" have same probability $1/2$. The entropy rate of such source is 1 bit/character [1], implying, that conventional sequential encoding of words $w_1, \ldots, w_m$ will cost at least mn bits.

Hereafter, we will often refer to an example shown Table I. In this case: $m = 8$, $n = 5$, and the total length $mn = 8 \times 5 = 40$ bits.

A. Tree-based representation

In order to construct a compact representation of the set $\{w_1, \ldots, w_m\}$, we employ a data structure, known as *digital search tree* or *DST* [3], [9], [10]. We start with a single root node, and assume that it corresponds to an empty word. We then pick our first word w_1, and depending on the value of its first character, we add left or right branch to the root node, and insert a new node there. We also store pointer to w_1 in that node. With second and subsequent words, we

TABLE I
EXAMPLE SET OF BINARY WORDS $\{w_1,\dots,w_m\}$.

Index i	**Word** w_i	**DST path** p_i	**Suffix** s_i
1	01011	0	1011
2	00111	00	111
3	10001	1	0001
4	01010	01	010
5	10010	10	010
6	00001	000	01
7	00110	001	10
8	00000	0000	0
Bits:	$8 \times 5 = 40$	18	22

traverse the tree starting from the root node, by following characters in a current word, and once we hit the leaf (a node with no continuation in the direction of interest), we extend it by creating a new node and storing pointer to the current word in it. This process is repeated m times, so that all words from our input set $\{w_1,\dots,w_m\}$ are inserted.

The DST structure constructed over our example set is shown in Fig.1. The paths from root to other nodes in the tree correspond to portions (prefixes) of words inserted in this structure. We list such prefixes in the third column in Table I. The forth column in Table I lists the remainders (suffixes) of each word. In other words, we observe that DST construction effectively "splits" words w_i $(i = 1,\dots,m)$ in two parts:

$$w_i = p_i\, s_i,$$

where p_i are prefixes covered by paths in the tree, and s_i are the remaining suffixes. Overall lengths of prefixes and the suffixes will be denoted by

$$P_m = \sum_{i=1}^{m} |p_i|, \quad \text{and} \quad S_m = \sum_{i=1}^{m} |s_i| = m\,n - P_m \qquad (1)$$

correspondingly. In our example, shown in Fig. 1, the overall DST path length is $P_m = 18$, and the length of the remaining suffixes is $S_m = 40 - 18 = 22$.

B. Encoding of a tree

Our next task is to encode the structure of the DST efficiently. More specifically, we need to encode the shape of its binary tree. This tree contains $i = m + 1$ nodes (m nodes associated with input words + root).

We start by scanning the tree recursively, by using pre-order tree traversal [4], and assigning labels "1" to the existing nodes, and "0" to missing ones (see Fig. 2). We call the resulting sequence of labels an *x-sequence*. It is known, that this sequence contains $2i + 1$ digits, and that it can serve as a unique representation of a tree with i nodes [2], [24]. Indeed, x-sequence may also serve as a code, but as we shall show, more compact representation is possible.

In general, it is known, that the total number of possible rooted binary trees with i nodes is given by the Catalan number [2, Section 2.3.4.4]:

$$C_i = \frac{1}{i+1}\binom{2i}{i}, \qquad (2)$$

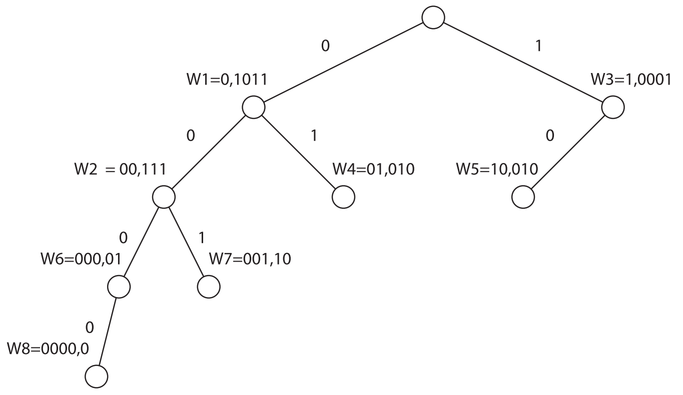

Fig. 1. Digital search tree (DST) structure constructed over our example set of words. We use commas to separate prefix (matching path from the root) and suffix parts of words.

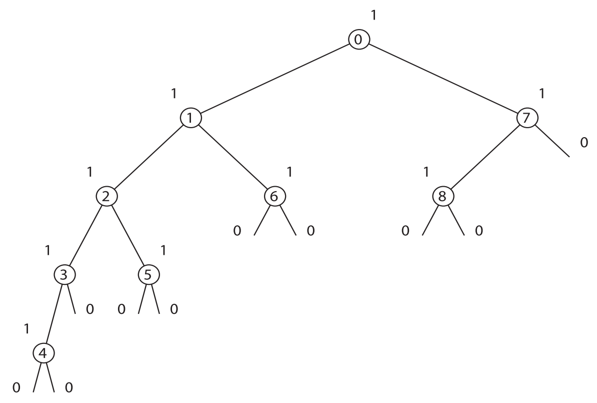

Fig. 2. Encoding of a binary tree. The numbers in nodes show order in which they are visited during pre-order tree traversal [4]. Present and missing nodes are labeled with 1's and 0's correspondingly. Scan of these labels produces a sequence: $x = 1111100010010011000$.

implying, that a tree can be uniquely represented by only

$$\lceil \log_2 C_i \rceil \sim 2\,i - \tfrac{3}{2}\log_2 i + O(1) \quad \text{[bits]}. \qquad (3)$$

We next briefly describe one possible coding technique [24] that achieves this rate.

Given an x-sequence for a tree, we produce a list of positions of symbols "1" in it. We will call it a z-sequence $z = z_1,\dots,z_i$. For example, for a sequence $x =$ 1111100010010011000, corresponding to a tree in Fig. 2, we produce: $z = 1, 2, 3, 4, 5, 9, 12, 15, 16$. We next define a rule for incremental reduction of z-sequences. Let j^* be the largest j, such that $z_j = j$. By $z^* = z^*_1,\dots,z^*_{i-1}$ we will denote a new sequence that omits value z_{j^*}, and subtracts 2 from all subsequent values in the original sequence:

$$z^*_j = \begin{cases} z_j, & j = 1,\dots,j^* - 1; \\ z_{j+1} - 2, & j \geqslant j^*. \end{cases}$$

Then, a lexicographic index (or *Zaks rank* Z) of a tree is recursively computed as follows:

$$\mathrm{Z}(z) = \begin{cases} 1, & \text{if} \quad j^* = i; \\ a_{i,j^*} + \mathrm{Z}(z^*), & \text{if} \quad j^* < i, \end{cases} \qquad (4)$$

where

$$a_{i,j} = \frac{j+2}{2i-j}\binom{2i-j}{i-j-1}, \quad 0 \leqslant j \leqslant i-1$$

are some constants [24].

For example, for a tree in Fig. 2, Zaks algorithm (4) produces:

$$\begin{aligned}
\mathrm{Z}(1,2,3,4,5,9,12,15,16) &= a_{9,5} + \mathrm{Z}(1,2,3,4,7,10,13,14);\\
\mathrm{Z}(1,2,3,4,7,10,13,14) &= a_{8,4} + \mathrm{Z}(1,2,3,5,8,11,12);\\
\mathrm{Z}(1,2,3,5,8,11,12) &= a_{7,3} + \mathrm{Z}(1,2,3,6,9,10);\\
\mathrm{Z}(1,2,3,6,9,10) &= a_{6,3} + \mathrm{Z}(1,2,4,7,8);\\
\mathrm{Z}(1,2,4,7,8) &= a_{5,2} + \mathrm{Z}(1,2,5,6)\\
\mathrm{Z}(1,2,5,6) &= a_{4,2} + \mathrm{Z}(1,3,4)\\
\mathrm{Z}(1,3,4) &= a_{3,1} + \mathrm{Z}(1,2)\\
\mathrm{Z}(1,2) &= 1;
\end{aligned}$$

resulting in

$$\begin{aligned}
&\mathrm{Z}(1,2,3,4,5,9,12,15,16)\\
&\quad= a_{9,5} + a_{8,4} + a_{7,3} + a_{6,3} + a_{5,2} + a_{4,2} + a_{3,1} + 1\\
&\quad= 154 + 110 + 75 + 20 + 14 + 4 + 3 + 1\\
&\quad= 381.
\end{aligned}$$

The code of this tree is simply a binary record of its index:

$$\mathrm{Bin}_{\lceil \log_2 C_{m+1} \rceil}(\mathrm{Z}) = \mathrm{Bin}_{13}(381) = 0000101111101.$$

As easily observed, this code is considerably shorter that $P_m = 18$ bits of prefix data stored in this tree.

We are now ready to describe the remaining steps in our coding scheme for sets of words.

C. Encoding and decoding algorithms

Given a set of m words $\{w_1,\dots,w_m\}$, our encoding algorithm performs the following operations:

1) Build, encode, and transmit DST structure over the input set $\{w_1,\dots,w_m\}$;
2) Traverse the tree and define a canonic order for nodes $i_1,\dots,i_m$ and prefixes $p_{i_1},\dots,p_{i_m}$ stored in the DST;
3) Encode and transmit suffixes according to canonic order: $s_{i_1},\dots,s_{i_m}$.

The construction of the DST structure and its encoding is performed as discussed in previous sections. To define a canonic order of nodes we use the standard pre-order tree traversal [4], and assign each node a serial number, starting with 0, assigned to the root node (see Fig. 3). As we reach a j-th node during the traversal, we can also recover prefix of a word w_{i_j} that was inserted in it. This produces an order $i_1,\dots,i_m$ in which prefixes of all words from our set can be retrieved from the tree. We omit the root node (and empty word that it contains) in this sequence. For example, for a tree in Fig. 3, this produces $i_1 = 1, i_2 = 2, i_3 = 6, i_4 = 8, i_5 = 7, i_6 = 4, i_7 = 3, i_8 = 5$. In order to transmit information about corresponding suffixes, we simply arrange and encode them in the same order: $s_{i_1},\dots,s_{i_m}$. Any standard source coding technique (such as Shannon, Huffman, or arithmetic codes) can be applied for this sequence.

The decoder performs the inverse sequence of operations:

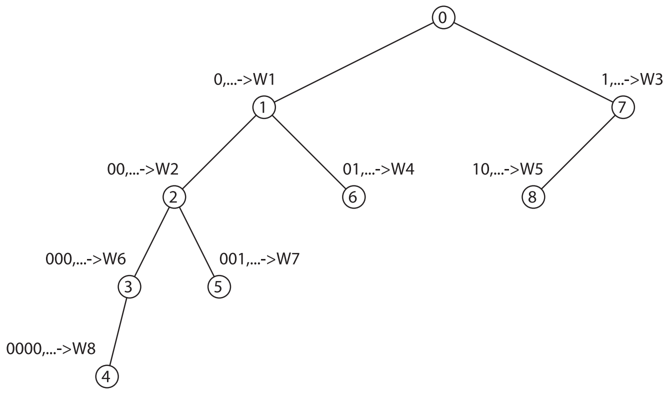

Fig. 3. Canonic order of nodes and words as they are encoded/decoded.

1) Decode the DST tree structure;
2) Traverse the tree, define canonic order of nodes, and retrieve corresponding prefixes $p_{i_1},\dots,p_{i_m}$;
3) Decode suffixes $s_{i_1},\dots,s_{i_m}$, and form complete decoded words: $w_{i_j} = p_{i_j}\, s_{i_j}$, $j = 1,\dots,m$.

We conclude our presentation by showing a complete DST-based code constructed for our example set of words (see Table 1, and Figures 1–3).

$$\begin{aligned}
&\mathrm{Code}(\{w_1,\dots,w_m\})\\
&\quad= \mathrm{Bin}_{\lceil C_{m+1} \rceil}(\mathrm{Z}), s_{i_1},\dots,s_{i_m}\\
&\quad= \mathrm{Bin}_{\lceil C_9 \rceil}(381), s_1, s_2, s_6, s_8, s_7, s_4, s_3, s_5\\
&\quad= 0000101111101, 1011, 111, 01, 0, 10, 010, 0001, 010.
\end{aligned}$$

As evident, the length of this code is $13 + 22 = 35$ bits, which is by $40 - 35 = 5$ bits shorter than the length of a straightforward sequential encoding of words in this set.

III. Analysis of Performance

Let us now assume that input words $\{w_1,\dots,w_m\}$ are produced by a general (not necessarily symmetric) binary memoryless source, emitting "0"s and "1"s with probabilities p and $q = 1 - p$ correspondingly. We assume that source parameter p is known. We further assume that all words are of length n, and that the total length of words in our set is

$$t = |w_1 \dots w_m| = m\, n\,. \tag{5}$$

If we apply a conventional code, such as Shannon or arithmetic code for a sequence of words $w_1,\dots,w_m$, then the *average length* of such code will satisfy:

$$\bar{L}_{\text{sequence}}(t) = H\, t + O(1), \tag{6}$$

where

$$H = -p \log_2 p - q \log_2 q, \tag{7}$$

is the entropy of the source [1].

We next estimate average length of our DST-based code.

Theorem 1. *The average length of DST-based code for a set of m binary words of length n, in a memoryless model satisfies (with $m,n \to \infty$, $n/\log_2 m > -\log_2^{-1} \max(p,q)$):*

$$\bar{L}_{\text{set}}(t) = H\, t - \log_2 m! + (A - \delta_1(m))\, m + O(\log m) \tag{8}$$

where A is a constant, and $\delta_1(m)$ is a zero-mean, oscillating function of a small magnitude. When $\log(p)/\log(q)$ is irrational, $\lim_{m\to\infty} |\delta_1(m)| = 0$. The exact value of A is

$$A = 2 - \frac{\gamma}{\ln 2} - \frac{H_2}{2\,H} + \alpha\,,$$

where $\gamma = 0.577\ldots$ is the Euler constant, H is the entropy of the source (7), $H_2 = p\log_2^2 p + q\log_2^2 q$, and

$$\alpha = -\sum_{k=1}^{\infty} \frac{p^{k+1}\log_2 p + q^{k+1}\log_2 q}{1 - p^{k+1} - q^{k+1}}.$$

Proof: We start by observing that condition $n/\log_2 m > -\log_2^{-1}\max(p,q)$ implies that the *height* (longest path) of DST constructed over m randomly generated input words is shorter than n [13]. This ensures that words in our set will be uniquely parsed by the tree.

Our code consists of 2 parts: (1) encoded DST structure, occupying at most $L_{\mathrm{DST}} = \lceil \log_2 C_{m+1} \rceil \leqslant \log_2 C_{m+1} + 1$ bits, and (2) encoded sequence of suffixes. We assume that Shannon code [1] is used to encode suffixes, producing at most $L_{\mathrm{suff}}(S_m) \leqslant H\,S_m + 1$ bits, where S_m is the total length of all suffixes, and H is the entropy of the source. Consequently, the expected code length becomes $\mathbf{E}L_{\mathrm{suff}}(S_m) \leqslant H\,\bar{S}_m + 1$, where $\bar{S}_m = \mathbf{E}S_m$, is the expected length of suffixes in our set. In turn, $\bar{S}_m$ can be expressed as $\bar{S}_m = t - \bar{P}_m$, where $\bar{P}_m = \mathbf{E}P_m$ is the expected path length in the DST tree.

We next retrieve result for the *expected path length* in DST [10]–[12]:

$$\bar{P}_m = \frac{m}{H}\left[\log_2 m + \frac{\gamma - 1}{\ln 2} + \frac{H_2}{2\,H} - \alpha + \delta_1(m)\right] + O(\log m)\,,$$

which introduces quantities H, H_2, γ, α and $\delta_1(n)$ appearing in the text of the theorem.

The rest becomes a matter of simple algebra:

$$\begin{aligned} \bar{L}_{\mathrm{set}} &= L_{\mathrm{DST}} + \mathbf{E}L_{\mathrm{suff}}(S_m) \\ &\leqslant \log_2 C_{m+1} + H\,\bar{\mathrm{S}}_m + 2 \\ &= \log_2 C_m + H\,\left(t - \bar{\mathrm{P}}_m\right) + 2 \end{aligned}$$

where by plugging expressions for $\bar{P}_m$ and C_m and subsequent simplifications we arrive at the expression claimed by the theorem. ■

By comparing average length of our code (8) with length of conventional code (6), we immediately notice about $\log_2 m!$ difference. The contribution of the following $O(m)$ term becomes relatively small as both n and m increase.

IV. Universal Codes for Sets of Codes

We now offer a modification of our scheme, allowing encoding of sets of words from unknown sources. As a component tool, we will use J. Ziv and A. Lempel's $LZ1$ code [19].

The proposed universal code for a set of m words $\{w_1,\ldots,w_m\}$, is constructed as follows:

1) Build, encode, and transmit DST structure over the input set $\{w_1,\ldots,w_m\}$;
2) Start $LZ1$ encoder, using our DST structure as initial dictionary;
3) Traverse the DST, and define a canonic order of nodes $i_1,\ldots,i_m$ and the corresponding prefixes $p_{i_1},\ldots,p_{i_m}$ stored in the DST;
4) Use $LZ1$ encoder to progressively encode and transmit all suffixes of words according to above defined order.

From the analysis of Lempel-Ziv algorithm (cf. [17], [19]), we know that the average length of $LZ1$ code for a sequence of words $w_1,\ldots,w_m$ in memoryless model satisfies:

$$\bar{L}^{LZ1}_{\text{sequence}}(t) = H\,t + H\big[B + \delta_2(t)\big]\frac{t}{\log_2 t} + O\left(t\frac{\log\log t}{\log^2 t}\right), \quad (9)$$

where $t = m\,n$ is the total input length, H is the entropy of the source, and B is a constant

$$B = \frac{2-\gamma}{\ln 2} - \frac{H_2}{2\,H} + \alpha,$$

where component quantities H_2, γ and α match ones described in Theorem 1, and $\delta_2(t)$ is another zero-mean fluctuating function of small magnitude.

We now estimate average length of the proposed DST-based universal code for sets of words.

Theorem 2. *The average length of universal DST-based code for a set of m binary words of length n, in a memoryless model satisfies (with $m,n\to\infty$, $n/\log_2 m > -\log_2^{-1}\max(p,q)$):*

$$\begin{aligned} \bar{L}^{LZ1}_{\text{set}}(t) \;\;\geqslant\;\; & H\,t - \log_2 m! + m\left[2 - \frac{2}{\ln 2} + \delta_3(m)\right] \quad (10) \\ & + H\left[B + \delta_2(t)\right]\frac{t}{\log_2 t} + O\left(t\frac{\log\log t}{\log^2 t}\right) \end{aligned}$$

where $\delta_2(t)$ and $\delta_3(m)$ are zero-mean, oscillating functions of small magnitude.

Proof: We first observe that DST structure for our set of input words $\{w_1,\ldots,w_m\}$ can always be converted into a sequence of prefixes $p_1,\ldots,p_m$. If LZ1 encoder is used to encode this sequence, it will construct a prefix tree, matching exactly the structure of the DST. The means, that by implementing a particular partition and reordering of our input words in a sequence we can produce $LZ1$ encoding that will coincide with ours after processing of first P_m symbols. Hence, in memoryless model, the length of our DST-based code can be expressed as:

$$\bar{L}^{LZ1}_{\text{set}}(t) = L_{\mathrm{DST}}(m) + \bar{L}^{LZ1}_{\text{sequence}}(t) - \mathbf{E}\bar{L}^{LZ1}_{\text{sequence}}(P_m)$$

where L_{DST} is the length of encoded DST structure, and $\mathbf{E}\bar{L}^{LZ1}_{\text{sequence}}(P_m)$ is the expected length of LZ1 codes constructed for a sequence of prefixes stored in DST.

We now observe that $\bar{L}^{LZ1}_{\text{sequence}}(t)$ is a "mostly" concave function of t. It becomes concave if we disregard small fluctuating function $\delta_2(t)$ in the $O\left(\frac{t}{\log t}\right)$ term and subsequent terms. By following this argument, and using Jensen's inequality we can conjecture that:

$$\mathbf{E}\bar{L}^{LZ1}_{\text{sequence}}(P_m) \leqslant \bar{L}^{LZ1}_{\text{sequence}}(\bar{P}_m) + O\left(\frac{\bar{P}_m}{\log \bar{P}_m}\right)$$

where $\bar{P_m} = \mathbf{E}P_m$ is the expected length of a all prefixes in the DST, and where $O\left(\frac{\bar{P}_m}{\log \bar{P}_m}\right)$ captures the magnitude of contribution of oscillating terms. More precise derivation is possible by applying analysis techniques used in [17], [18].

By combining (IV) and (IV) together, plugging in expressions for P_m and $\bar{L}^{LZ1}_{\text{sequence}}$, and some simple algebra, we arrive at expression claimed by the theorem. ■

Again, by comparing estimated length of our code (10) with the length of LZ1 code (9), we notice about $\log_2 m!$ difference. The contribution of the following $O(m)$ term becomes relatively small as both n and m increase.

In passing, we must note that the use of LZ1 code offers just one possible way of achieving universality for codes for sets. It is simple, but it may not be the best. For example, for a class of memoryless sources, the use of Krichevsky-Trofimov codes [32] is known to be more efficient. Their use should reduce the $O\left(\frac{t}{\log t}\right)$ redundancy term in (10) to just $O(\log t)$.

V. Conclusions

A simple construction procedure for design of codes for unordered sets of words is offered. The performance of proposed codes is analyzed, and it is shown that in the memoryless model such codes offer close to $\log m!$ (where m is the number of words) rate savings compared to sequential encoding of same words. It is shown that such codes can be designed for both known and unknown sources.

References

[1] T. M. Cover and J. M. Thomas, *Elements of Information Theory*, John Wiley & Sons, New York, 1991.

[2] D. Knuth, *The Art of Computer Programming. Fundamental Algorithms. Vol. 1*, Addison-Wesley, Reading MA, 1968.

[3] D. Knuth, *The Art of Computer Programming. Sorting and Searching. Vol. 3*, Addison-Wesley, Reading MA, 1973.

[4] R. Sedgewick, *Algorithms. Parts 1-4. Fundamentals, Data Structures, Sorting, Searching*, Addison-Wesley, Reading MA, 1998.

[5] A. Lempel, On multiset decipherable codes, *IEEE Trans. Inf. Theory* vol. 32, no. 5, pp. 714–716, 1986.

[6] S. A. Savari, Compression of words over a partially commutative alphabet, *IEEE Trans. Inf. Theory*, vol. 50, no. 7, pp. 1425–1441, 2004.

[7] L. R. Varshney and V. K. Goyal, Toward a Source Coding Theory for Sets, *Proc. Data Compression Conference*, Snowbird, Utah, March 2006, pp. 13–22.

[8] Y. A. Reznik, Coding of Sets of Words, *Proc. Data Compression Conference*, Snowbird, Utah, April 2011, pp. 43–52.

[9] E. G. Coffman, Jr. and J. Eve, File structures using hashing functions, *Communications of the ACM*, vol. 13, no. 7, pp. 427–436, 1970.

[10] P. Flajolet and R. Sedgewick, Digital Search Trees Revisited, *SIAM J. Computing*, vol. 15, pp. 748–767, 1986.

[11] P. Kirschenhofer and H. Prodinger, Some further results on digital search trees, *Lecture Notes in Computer Science*, vol. 229, pp. 177–185, Springer-Verlag, New York, 1986.

[12] W. Szpankowski, A characterization of digital search trees from the successful search viepoint, *Theoretical Computer Science*, vol. 85, pp. 117–134, 1991.

[13] B. Pittel, Asymptotic growth of a class of random trees, *Annals of Probability*, vol. 18, pp. 414–427, 1985.

[14] A. Andersson and S. Nilsson, Improved Behaviour of Tries by Adaptive Branching, *Information Processing Letters*, vol. 46, pp. 295–300, 1993.

[15] Y. A. Reznik, Some Results on Tries with Adaptive Branching, *Theoretical Computer Science*, vol. 289, no. 2, pp. 1009–1026, 2002.

[16] Y. A. Reznik, On the Average Depth of Asymmetric LC-tries, Information Processing Letters, vol. 96, no. 3, pp. 106–113, 2005.

[17] G. Louchard and W. Szpankowski, On the Average Redundancy Rate of the Lempel-Ziv Code, *IEEE Trans. Information Theory*, vol. 43, pp. 2–8, 1997.

[18] Y. A. Reznik and W. Szpankowski, On the Average Redundancy Rate of the Lempel- Ziv Code with the K-Error Protocol, *Information Sciences*, vol. 135, pp.57–70, 2001.

[19] J. Ziv and A. Lempel, Compression of Individual Sequences via Variable-Rate Coding, *IEEE Trans. Information Theory*, vol. 24, pp. 550–536, 1978.

[20] B. P. Tunstall, *Synthesis of Noiseless Compression Codes*, Ph.D. dissertation, Georgia Inst. Tech., Atlanta, GA, 1967.

[21] G. L. Khodak, Redundancy Estimates for Word-Based Encoding of Messages Produced by Bernoulli Sources, *Probl. Inform. Trans.*, vol. 8, no. 2, pp. 21–32, 1972. (in Russian)

[22] M. Drmota, Y. A. Reznik, S. A. Savari, and W. Szpankowski, Precise Asymptotic Analysis of the Tunstall Code, *Proc. IEEE Intl. Symp. Inf. Theory (ISIT06)*, Seattle, USA, 2006, pp. 2334 – 2337.

[23] F. Williams, Y. Shtarkov, T. Tjalkens, The Context-Tree Weighting Method: Basic Properties, *IEEE Trans. Inform. Theory*, vol. 41, n. 3, pp. 653–664, 1995.

[24] S. Zaks, Lexicographic Generation of Ordered Trees, *Theoretical Computer Science*, vol. 10, 1980, pp. 63–82.

[25] J. Katajanen and E. Makinen, Tree compression and optimization with applications, *International Journal of Foundations of Computer Science*, vol. 1, no. 4, 1990, pp. 425–447.

[26] E. Makinen, A survey of binary tree codings, *The Computer Journal*, vol. 34, no. 5, 1991, pp. 438–443.

[27] B. Ryabko, Fast enumeration of combinatorial objects, *Discrete Math.and Applications,* vol. 10, no. 2, pp. 163–182, 1998.

[28] D. Lewis, Naive (Bayes) at Forty: The Independence Assumtion in Information Retrieval, *Proc. 10th European Conference on Machine Learning (ECML-98)*, 1998, pp. 4-15.

[29] T. Gagie, Compressing Probability Distributions, *Information Processing Letters*, vol. 97, no. 4, pp. 133–137, 2006.

[30] D. Chen, S. Tsai, V. Chandrasekhar, G. Takacs, J. Singh, and B. Girod, Tree histogram coding for mobile image matching, *in Proc. IEEE Data Compression Conference*, Snowbird, Utah, March 2009, pp. 143-153.

[31] V. Chandrasekhar, G. Takacs, D. Chen, S. Tsai, Y. Reznik, R. Grzeszczuk, and B. Girod, Compressed Histogram of Gradients: a Low-bitrate Descriptor, *International Journal of Computer Vision*, vol. 94, pp. 1-16, 2011.

[32] R. E. Krichevsky and V. K. Trofimov, The Performance of Universal Encoding, *IEEE Trans. Information Theory*, **27** (1981) 199–207.