Coding of 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 natural language processing, retrieval, machine learning, computer vision, and other fields. We note that this problem is different from the problem of coding of a particular sequence of same words, and show that up to $\log_2 m!$ bits (where m is the number of words in the set) can be saved by specialized codes for sets. We propose one possible design of such codes, and prove its asymptotic optimality in the memoryless model.

I. Introduction

The classic problem of source coding is to encode a given *sequence of words* (or a *message*) $w = w_1, w_2, \ldots$, with the goal of minimizing the number of bits needed for such encoding. 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 and techniques, 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. Here we need something else.

The purpose of this paper is to offer one possible practical solution for 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] (cf. [3], [10]). It is similar, in a sense, to the *prefix tree* or *incremental parsing rule* of J. Ziv and A. Lempel's *LZ 78* algorithm [19]. However, prefix tree always parses a single sequence, while DST is designed to parse a set of sequences. DST is also different from parsing trees used by Tunstall 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 were 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 it asymptotically approaches the expected $\log_2 m!$ reduction in the bitrate.

Among related prior studies, we must mention 1986 work of A. Lempel [6], who defined a class 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 [7]. The achievable performance bounds for coding of sets were studied by L. Varshney and V. Goyal [8]. The rate reduction limit of $\log_2 m!$ bits in

Table I
EXAMPLE SET OF BINARY WORDS $\{w_1, \ldots, w_m\}$.

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

lossless regime was also obtained in [8], but without offering any constructive process for reaching it. 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 [28], [29], [34].

This paper is organized as follows. In Section II we provide detailed description of our algorithm. In Section III, we study asymptotic performance of this scheme in the memoryless model. Section IV contains discussion on practical relevance of this work, its applications, and conclusions.

II. CODING OF SETS OF WORDS

Let $\{w_1, \ldots, w_m\}$ be a set of words that we need to encode. For simplicity of presentation, we will assume that these words are binary, distinct, have the same length $|w_i| = n$, and produced by a *symmetric memoryless source*. That is, characters "0" and "1" appear with same probability $p = 1 - p = 1/2$ regardless of their positions or order. 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 set of words shown Table I (second column). In this case: $m = 8$, $n = 5$, and total length $mn = 8 \times 5 = 40$ bits.

A. Tree-based representation

In order to construct a more 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 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 $\{w_1, \ldots, w_m\}$ are inserted.

The DST structure constructed over our example set is shown in Figure 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 ($i = 1, \ldots, m$) in two parts:

$$w_i = p_i \, s_i,$$

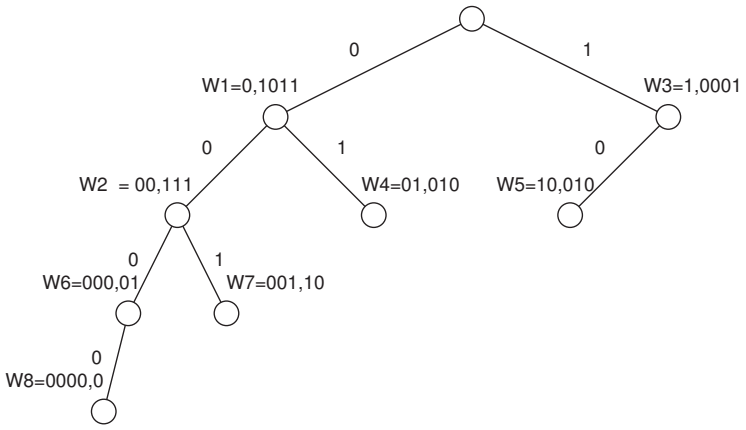

Figure 1. Digital search tree (DST) structure constructed over our example set of words. We use commas to separate prefix (matching path from the root) and suffix parts of words.

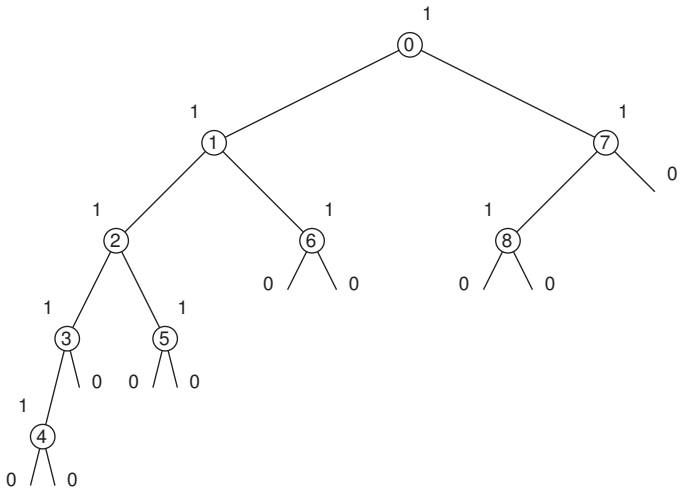

Figure 2. Encoding of a binary tree. The numbers in nodes show order in which they are visited during pre-order tree traversal [4]. Present and missing nodes are labeled with 1's and 0's correspondingly. Scan of these labels produces a sequence: $x = 1111100010010011000$.

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

$$\mathrm{P}_m = \sum_{i=1}^{m} |p_i|, \quad \text{and} \quad \mathrm{S}_m = \sum_{i=1}^{m} |s_i| = m\,n - P_m \tag{1}$$

correspondingly. In our example, shown in Figure 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 Figure 2). We call the resulting sequence of

Table II
COEFFICIENTS $a_{i,j}$ USED IN LEXICOGRAPHIC ENUMERATION OF TREES.

i \ j	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
1	1								
2	2	1							
3	5	3	1						
4	14	9	4	1					
5	42	28	14	5	1				
6	132	90	48	20	6	1			
7	429	297	165	75	27	7	1		
8	1430	1001	572	275	110	35	8	1	
9	4862	3432	2002	1001	429	154	44	9	1

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]:

$$C_i = \frac{1}{i+1}\binom{2i}{i}, \tag{2}$$

implying, that a tree can be uniquely represented by only

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

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 $z = z_1, \ldots, z_i$. For example, for a sequence $x = 1111100010010011000$, corresponding to a tree in Figure 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 $z^* = z_1^*, \ldots, z_{i-1}^*$ we will denote a new sequence that omits value z_{j^*}, and subtracts 2 from all subsequent values in the original sequence:

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

Then, a lexicographic index (or *Zaks rank*) of a tree is recursively computed as follows [24]:

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

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 (see Table II).

For example, for a tree in Figure 2, Zaks ranking algorithm (4) produces:

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

resulting in

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

The code of this tree is a $\lceil \log_2 C_{m+1} \rceil = \lceil \log_2 C_9 \rceil = 13$ bits-long binary record of its index:

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

As easily observed, this code is shorter that $2(m+1)+1 = 19$ bits used by our x-sequence, and notably, it is also shorter than $P_m = 18$ 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. Coding of sets of words

Given a set of m words $\{w_1, \ldots, w_m\}$, the proposed algorithm performs the following operations:

1) Build, encode, and transmit DST structure over the input set $\{w_1, \ldots, w_m\}$;
2) Scan the tree recursively, and define a canonic order of nodes and the corresponding prefixes $p_{i_1}, \ldots, p_{i_m}$ in the DST;
3) Encode and transmit suffixes according to same order $s_{i_1}, \ldots, s_{i_m}$.

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 Figure 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 $i_1, \ldots, i_m$ 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 Figure 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: $s_{i_1}, \ldots, s_{i_m}$. Any standard source coding technique (such as Shannon, Huffman, or arithmetic codes) can be applied for this sequence.

The decoder performs inverse operations:

1) Decode the DST tree structure;
2) Scan nodes in the same order as encoder, and retrieve prefixes $p_{i_1}, \ldots, p_{i_m}$;
3) Sequentially decode corresponding suffixes $s_{i_1}, \ldots, s_{i_m}$, and form complete decoded words: $w_{i_j} = p_{i_j}\, s_{i_j}$, $j = 1, \ldots, m$.

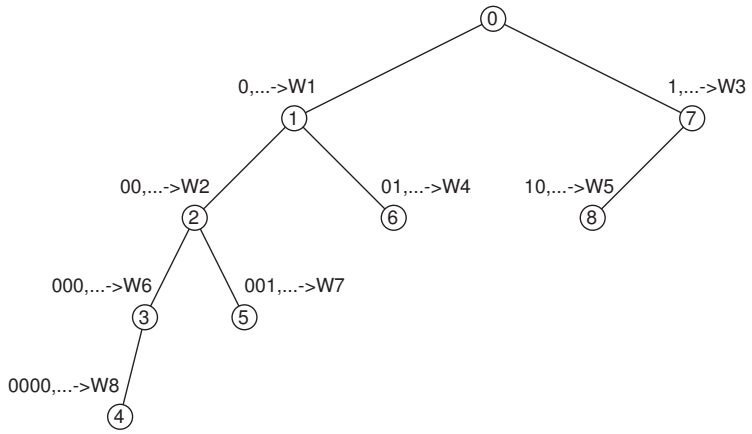

Figure 3. Canonic order of nodes (produced by pre-order tree-traversal, with count 0 assigned to root), corresponding prefixes, and words from our example set.

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

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

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 $\{w_1,\ldots,w_m\}$ 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.

Recall, that the entropy rate of memoryless source is given by [1]

$$h(p) = -p\log_2 p - q\log_2 q, \tag{5}$$

and so the ideal *average length* of encoding of a sequence $w_1,\ldots,w_m$ becomes

$$\mathrm{L}^*_{\mathrm{sequence}}(mn,p) = mn\,h(p) \quad [\mathrm{bits}], \tag{6}$$

where n denotes the length of each word, and mn is the length of the entire sequence.

If we now allow arbitrary reordering of the decoded words in the set $\{w_1,\ldots,w_m\}$, we may expect the ideal average length of such code to become:

$$\mathrm{L}^*_{\mathrm{set}}(m,n,p) = m\,n\,h(p) - \log_2 m! \quad [\mathrm{bits}]. \tag{7}$$

As obvious, we must also require that $\mathrm{L}^*_{\mathrm{set}}(m,n,p) > 0$, and therefore we will expect word length n to be sufficiently large. In the asymptotic sense, with $m,n\to\infty$, this implies that:

$$\frac{n}{\log_2 m} > \frac{1}{h(p)}.$$

Given a specific algorithm ξ producing codes with average lengths $\mathrm{L}_\xi\,(m,n,p)$, we will define its *average redundancy rate for coding of sets* as follows:

$$\begin{aligned}\mathrm{R}_\xi(m,n,p) &= \frac{1}{m\,n}\left[\mathrm{L}_\xi\,(m,n,p) - \mathrm{L}^*_{\mathrm{set}}\,(m,n,p)\right] \\ &= \frac{1}{m\,n}\,\mathrm{L}_\xi\,(m,n,p) - \left[h(p) - \frac{1}{m\,n}\,\log_2 m!\right]. \end{aligned} \tag{8}$$

This definition is almost identical to one used in traditional source coding [5], except that in our case, the ideal rate is no longer the entropy $h(p)$, but rather $h(p) - \frac{1}{m\,n}\,\log_2 m!$.

Now, before we can introduce our main result, we need to make one more clarification. In our analysis, we will assume, that $m, n \to \infty$, and moreover, that

$$\frac{n}{\log_2 m} > \frac{1}{-\log_2 \max(p,q)}\,. \tag{9}$$

This condition says that with probability 1, the height (longest path) of DST constructed over m randomly generated input words $w_1, \ldots, w_m$ will be shorter than n, or equivalently, that length n will be sufficient to uniquely parse all input words by a DST structure (cf. [13]). Since $-\log_2 \max(p,q) \leqslant h(p)$, condition (9) also ensures that rate limit (7) is positive.

The main result of analysis of our proposed coding scheme is given below.

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

$$\mathrm{R}_{\mathrm{DST}}\,(m,n,p) = \frac{1}{n}\left[A(p) + \delta(m) + O\left(\frac{\log m}{m}\right)\right] \tag{10}$$

where $A(p)$ is a constant, which exact value is given by:

$$A(p) = 2 - \frac{\gamma - 2}{\ln 2} - \frac{h_2(p)}{2\,h(p)} + \alpha(p) \tag{11}$$

where: $\gamma = 0.577\ldots$ is the Euler constant, $h(p)$ is the entropy of the source (5),

$$h_2(p) = p\log_2^2 p + q\log_2^2 q,$$

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

and $\delta(m)$ is a zero-mean, oscillating function of a small magnitude.

Proof: We know that our code consists of 2 parts: 1) encoded tree, occupying at most $\lceil \log_2 C_{m+1} \rceil \leqslant \log_2 C_{m+1} + 1$ bits, and 2) encoded sequence of suffixes. We will assume that block Shannon code [1] is used to encode the suffixes. This produces code with the following expected length:

$$L_{\mathrm{suff}}(S_m, p) \leqslant S_m\,h(p) + 1\,,$$

where $S_m = m\,n - D_n$ is the total length of all suffixes, and $h(p)$ is the entropy of the source.

Since $L_{\mathrm{suff}}(S_m, p)$ is bounded by a linear function of S_m, we can further expect that the average length of code, considering all possible suffix lengths, will be

$$\sum_s \Pr(S_m = s)\; L_{\mathrm{suff}}(s,p) \leqslant \bar{S}_m\,h(p) + 1\,,$$

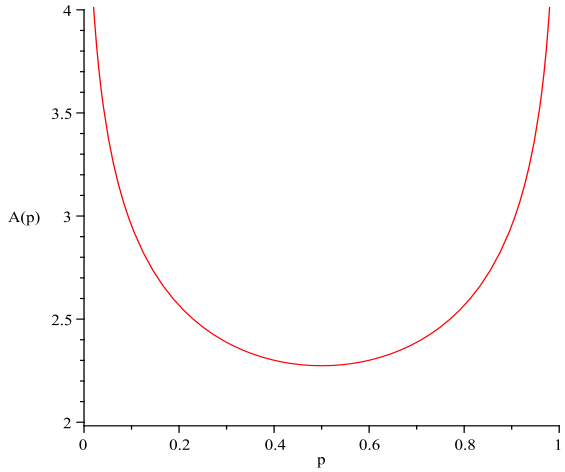

Figure 4. Behavior of redundancy factor $A(p)$ as function of parameter of the source p.

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 $\bar{S}_m = m\,n - \bar{P}_n$, where $\bar{P}_m = \mathbf{E}P_m$ is the expected path length in the DST tree.

We next retrieve result for so-called *average depth* of the DST [10]–[12]:

$$\frac{1}{m}\bar{P}_m = \frac{1}{h(p)}\left[\log_2 m + \frac{h_2(p)}{2\,h(p)} + \frac{\gamma - 1}{\ln 2} - \alpha + \delta_1(m) + O\left(\frac{\log m}{m}\right)\right],$$

which introduces quantities $h(p)$, $h_2(p)$, γ, $\alpha(p)$ and $\delta_0(n)$ appearing in the text of our theorem.

The rest becomes a matter of simple algebra. Expected code length turns into

$$\begin{aligned} L_{\mathrm{DST}}\,(m,n,p) &\leqslant \log_2 C_{m+1} + \bar{\mathrm{S}}_m\,h(p) + 2 \\ &= mn\,h(p) + \log_2 C_{m+1} - \bar{\mathrm{P}}_m\,h(p) + 2 \\ &= \dots \\ &= mn\,h(p) - m\log_2 m + m\left[2 - \frac{\gamma-1}{\ln 2} - \frac{h_2(p)}{2\,h(p)} + \alpha(p) - \delta(m)\right] + O(\log m) \end{aligned}$$

and by combining this with asymptotic expansion of

$$\log_2 m! = m\log_2 m - \frac{1}{\ln 2}m + O\left(\log m\right)$$

we arrive at the redundancy term claimed by the theorem. ■

A. Discussion

As evident, the average redundancy rate of our proposed scheme (10) decays at rate $O\left(\frac{1}{n}\right)$ as word length n increases. This is a good sign: it matches the order of best attainable redundancy rate when encoding a single n-bit long sequence from a known source [5].

It is also encouraging that the redundancy rate is not growing with the number of words in our set m. Recall, that if we would use simple sequential encoding, this would cause $\log_2 m!$ overhead, and so we would observe a logarithmic ($\frac{\log_2 m!}{mn} \sim \frac{\log_2 m}{n}$) increase of redundancy rate with m. In our case, the redundancy rate stays almost constant w.r.t. m, defined by the leading factor $A(p)$ and a small-magnitude oscillating function $\delta(m)$. We provide plot of values of factor $A(p)$ in Figure 4.

Overall, this analysis shows that the proposed scheme delivers close to the predicted optimal performance in the memoryless model (7), and that the difference (redundancy rate) becomes progressively small, as the length of words n increases.

IV. Applications, Significance, and Concluding Remarks

The representation of a text by an unordered collection of words is a common simplifying assumption [27] used in many fields, including natural language processing, retrieval, document classification, machine learning, and computer vision. It is usually called the "bag of words" (BoW) or "bag of features" (BoF) representation, and it serves as a basic element for further processing. When such processing is done remotely, for example by a search / classification engine located in the cloud, bags of words may need to be transmitted over the network to initiate search / classification processing. The use of compression technique discussed in this paper may allow more efficient transmission or storage of such data.

Examples of existing practical applications that can immediately benefit from this work include *mobile visual search* and *mobile augmented reality* applications [30], [31]. Such applications usually initiate search by taking a picture of an object of interest, then perform extraction of a set of visual words, for example SIFT, SURF, or CHoG images features [32]–[35], and then use this set for retrieval. Typically, each query image becomes represented by approximately $m \sim 1000$ features, therefore, about $\log_2 m! \sim 8.3$K bits per each query request can be saved. With compact image features, such as CHoG [34], [35] this could lead to practically appreciable improvements in performance.

In general, the larger is the number of words in the BoF representation, the more significant is the effect of using specialized encoding for unordered sets. The asymptotic relationship between the expected rate reduction factor $\xi = \frac{\log_2 m!}{mnh(p) - \log_2 m!}$ and the size of the set m boils down to:

$$m \sim 2^{nh(p)\frac{\xi}{1+\xi}}.$$

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] R. E. Krichevsky, *Universal Data Compression and Retrieval*, Kluwer, Norwell, MA, 1993.

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

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

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

[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] 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.

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

[29] 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.

[30] G. Takacs, V. Chandrasekhar, N. Gelfand, Y. Xiong, W.-C. Chen, T. Bismpigiannis, R. Grzeszczuk, K. Pulli, and B. Girod, Outdoors Augmented Reality on Mobile Phone using Loxel-Based Visual Feature Organization, *Proc. ACM International Conference on Multimedia Information Retrieval (MIR)*, 2008.

[31] B. Girod, V. Chandrasekhar, D. Chen, N-M. Cheung, R. Grzeszczuk, Y. Reznik, G. Takacs, S. Tsai, and R. Vedantham, Mobile Visual Search, *IEEE Signal Processing Magazine* - to appear.

[32] D. Lowe, Distinctive Image Features from Scale-Invariant Keypoints, *International Journal of Computer Vision*, vol. 60, no. 2, pp. 91-110, 2004.

[33] H. Bay, A. Ess, T. Tuytelaars, L. Van Gool, SURF: Speeded Up Robust Features, *Computer Vision and Image Understanding*, vol. 110, no. 3, pp. 346359, 2008.

[34] V. Chandrasekhar, G. Takacs, D. Chen, S. Tsai, R. Grzeszczuk, B. Girod, CHoG: Compressed Histogram of Gradients – A low bit-rate feature descriptor, *Proc. Computer Vision and Pattern Recognition (CVPR09)*, pp. 2504-2511, 2009.

[35] V. Chandrasekhar, Y. Reznik, G. Takacs, D. Chen, S. Tsai, R. Grzeszczuk, and B. Girod, Quantization Schemes for Low Bitrate Compressed Histogram of Gradient Descriptors, *Proc. Computer Vision and Pattern Recognition (CVPR10)*, 2010.