ISIT'2013 1569718387

Uniquely Decodable and Directly Accessible Non-Prefix-Free Codes via Wavelet Trees

M. Oğuzhan Külekci
TÜBİTAK - BİLGEM National Research Institute of Electronics and Cryptology, Gebze, Kocaeli, Turkey
Email: oguzhan.kulekci@tubitak.gov.tr

***Abstract*—Unique decodability is the essential feature of any coding scheme, which is naturally provided by prefix–free codes satisfying the Kraft-McMillan inequality. Non–prefix–free codes have received much less attention due to the lack of an efficient method to support this property. In this study we introduce a novel technique that uses wavelet trees to bring unique decodability to non–prefix–free codes. Proposed method also provides direct access to the ith codeword, which can be extended to any variable-length coding scheme. The space overhead required for unique decoding is upper bounded by $n \cdot \log q$ bits, where n is the number of symbols, and q is the number of distinct codeword lengths, which is normally expected to be a small number in non-prefix-free codes. Direct access is supported by using an additional $o(n \cdot \log q)$ bits. We show that the overhead space requirement is much less than sampling methods using state-of-the-art compact integer representations.**

I. Introduction

Wavelet trees, introduced by Grossi et al. [1], rapidly became a fundamental data structure in the last decade and found applications in many areas [2], [3], [4]. In this study we investigate yet another usage of it, particularly in *coding*. We show that wavelet trees can efficiently provide (i) unique decodability of non-prefix-free codes, and (ii) direct accessibility in any variable–length coding scheme.

Unique decodability is the essential feature of any coding scheme to be able to extract the original sequence from the encoded stream without any loss or ambiguity. Decoding the original symbols in fixed–length coding is trivial as the number of bits reserved for every codeword is equal, and the codewords are distinct. The ith codeword encoding the ith symbol can easily be reached just by moving forward $(i-1) \cdot K$ bits, where K is the fixed codeword length.

Variable-length codes [5] require some more sophisticated conditions to hold for unique decoding as (i) none of the codewords should be a prefix of another codeword, and (ii)the lengths of the codewords must satisfy the Kraft's [6] inequality. The coding schemes supporting these properties are uniquely decodable prefix-free codes.

In variable–length coding, one needs to decode all previous $(i-1)$ symbols to access to the ith symbol since the code-lengths are variable. Sampling, which stores some (*sparse*) or all (*dense*) of the codeword boundaries on the code-stream helps to reach the ith symbol in better time with the cost of increased space usage.

Recently, Brisaboa *et al.* [7] introduced an elegant solution for Vbyte [8] variable length coding scheme, which does not bring a heavy overhead and provides fast direct access to any symbol in the sequence. However, their technique is only applicable particularly to Vbyte coding, and not a general solution for the direct accessibility of variable–length codes. To the best of our knowledge, no work other than the sampling methods generalizing direct access to variable-length codes has appeared yet. Beyond unique decodability of non–prefix–free codes, the solution we introduce in this study fills this gap and generalizes direct access for variable–length codes.

Prefix-free codes, particularly in data compression, have been investigated deeply aiming to represent an input sequence in space as close as to its entropy by removing the redundancies. On the other hand, although non-prefix-free codes might be able to provide better performance in terms of compression, they have not received much attention despite some limited studies [9] due to the lack of an efficient technique providing unique decodability.

Since in non–prefix–free coding some codewords may be prefixes of some others, ambiguities are not avoidable and normally arise while decoding the code-stream. It is discussed in [9] that it is possible to avoid those ambiguities by considering the transactions between the symbols that cause ambiguous parses might be forbidden on the original sequence. However, expecting such limitations to hold especially on large alphabets is optimistic, and seems difficult to be used in practice.

In this study, we introduce a novel method that uses wavelet trees to bring unique decodability to non–prefix–free codes. We also show that the proposed method supports direct access. Although we mainly aim to provide a solution for non–prefix–free codes, any variable length coding scheme can benefit from the introduced technique to support direct accessibility.

The overhead space consumption of the proposed method is upper bounded by $n \cdot \log q$ bits to support unique decodability, where q is the number of distinct codeword lengths. By using an additional $o(n \cdot \log q)$ bits, it is also possible to support direct accessibility. The parameter q is expected to be around $o(\log \sigma)$, where σ is the source alphabet size. Thus, the overhead space complexity is around $O(n \cdot \log \log \sigma)$ with a constant factor of less than 2.

The paper is organized as follows. In section 2 we formally define the problem and introduce the notation used throughout the study. In section 3 we analyze possible solutions based on the previous related works. We describe our method in section 4, which is followed by the section we discuss and give comparison against alternatives. Conclusions summarize the

contributions and address some future research opportunities.

II. The Problem and the Notation

$T = t_1 t_2 \dots t_n$ is a sequence of n symbols, where each t_i, $1 \le i \le n$, is drawn from alphabet $\Sigma = \{\epsilon_1, \epsilon_2, \dots, \epsilon_\sigma\}$. The coding scheme $\mathcal{C} : \Sigma \to \mathcal{A}$ maps symbols of Σ to binary codewords in $\mathcal{A} = \{\alpha_1, \alpha_2, \dots, \alpha_\sigma\}$ such that $\mathcal{C}(\epsilon_i) = \alpha_i$.

$\mathcal{C}$ is a variable–length scheme when the lengths of the codewords, $|\alpha_i|$s, are not all equal. If, for some i and j, the codeword α_i is a prefix of α_j, then $\mathcal{C}$ is *non–prefix–free*. We assume all codewords are distinct in $\mathcal{A}$, and set $L = \{\ell_1, \ell_2, \dots, \ell_q\}$ represents the unique codeword lengths observed in $\mathcal{A}$.

The encoding of T is $\mathcal{C}(T) = \mathcal{C}(t_1)\mathcal{C}(t_2)\dots\mathcal{C}(t_n)$, where $\forall i$, $\mathcal{C}(t_n) \in \mathcal{A}$. For the sake of clarity we replace $\mathcal{C}(t_i)$ by c_i, and thus represent $\mathcal{C}(T) = c_1 c_2 \dots c_n$. The total length of the code-stream $\mathcal{C}(T)$ in bits is represented by $|\mathcal{C}(T)| = |c_1| + |c_2| + \dots + |c_n|$, where $|c_i|$ is the individual length of the codeword corresponding to symbol t_i.

When $\mathcal{C}$ is non–prefix–free, the correct decoding of $\mathcal{C}(T)$ is not always guaranteed as there might be more than one possible parse of the encoded sequence. For example, if α_i is a prefix of α_j, $\alpha_i\alpha_a$ and $\alpha_j\alpha_b$ might have the same bit representation for some α_a and α_b, which causes ambiguities during decoding. To overcome this problem and guarantee unique decodability of $\mathcal{C}$, we need to know the indices of the codeword boundaries on $\mathcal{C}(T)$. Assuming the beginning bit position of each codeword c_i in $\mathcal{C}(T)$ is shown by p_i, the set $P = \{p_1, p_2, \dots, p_{n-1}\}$ should be attached to code-stream $\mathcal{C}(T)$ to provide unique decodability. We will investigate various compact representation of P in the next section.

Another problem we consider is the difficulty of accessing a random codeword c_i in $\mathcal{C}(T)$. In variable–length coding, since the codeword lengths are not all equal, the calculation of the beginning bit address of c_i either requires decoding all previous codewords $c_1 c_2 \dots c_{i-1}$, or storing some tap points among $\mathcal{C}(T)$, which are the sampled codeword boundaries, and perform decoding previous symbols not from c_1, but from the nearest sampled position. Note that the latter case is a time–memory trade–off, where access time is improved by using extra space to store more sampled positions. Direct access to c_i is only possible if we use a dense sampling and record every item in P, which is much more space consuming.

III. Related Studies and Alternative Solutions

The naive way to provide unique decodability along with direct access in a non-prefix-free coding scheme is to store set P, which requires $n \cdot \log|\mathcal{C}(T)|$ bits. However, benefiting from recent advances in compact integer representations [10], there are more efficient ways of that storage both in terms of time and space complexity.

The *dense sampling* introduced by Ferragina & Venturini [11] can be used to mark the codeword boundaries on $\mathcal{C}(T)$. In FV scheme, the pointers to codeword beginning positions are maintained in a two–level structure such that the address of c_i is computed by first locating to an absolute position on the encoded stream and then reaching the target position by moving relatively to that absolute address. Hence, each pointer is composed of an absolute address and a relative address. Selecting the super block size around $\Theta(|\mathcal{C}(T)|)$, this requires an overhead of $O\big(n{\cdot}(\log\log|\mathcal{C}(T)|{+}\log\log max(L))\big)$. where $max(L) = max(|\alpha_1|, |\alpha_2|, \dots, |\alpha_\sigma|)$ is the longest codeword length in $\mathcal{A}$. The address of any codeword can be computed in constant time, and the length of the codeword is simply the difference between starting positions of consecutive items. Thus, dense sampling of FV over $\mathcal{C}(T)$ supports direct access.

Method	Overhead space complexity in bits
naive	$n \cdot \log\|\mathcal{C}(T)\|$
Ferragina & Venturini [11]	$O\big(n(\log\log\|\mathcal{C}(T)\| + \log\log max(L))\big)$
Brisaboa *et al.* [7]	$\Upsilon + \frac{\Upsilon}{b} + o(\frac{\Upsilon}{b})$
Elmasry *et al.* [12] (also by Delpratt *et al.* [13])	$n \cdot \log(\frac{\|\mathcal{C}(T)\|}{n}) + O(n)$
this paper	$n \cdot \log q + o(n \cdot \log q)$

TABLE I
Additional space requirements of possible alternatives to uniquely decode a non-prefix-free coding scheme. $\mathcal{C}(T)$ is the length of the code-stream in bits, n is the total number of symbols encoded, L is the set of distinct codeword lengths, q represents the size of L, and $\Upsilon = \sum_{\forall i}\lfloor\log p_i\rfloor = \sum_{i=1}^{n-1}\sum_{j=1}^{j\le i}|c_j|$.

Brisaboa *et al.* [7] recently proposed a very efficient technique, namely DAC, both in time and space for compact representation of integer sequences that support direct access. They represent the integers via (an enhanced version of) *Vbyte* [8] variable length coding scheme. In Vbyte coding the minimal representation of an integer x ($\lceil\log x\rceil$ bits) is split into blocks of b bits, where each block is attached with one bit that is set to 1 on the final block and 0 otherwise. Direct random access is provided by collecting the marker bits of blocks into separate arrays, on which constant time binary rank/select queries are available by reserving some additional space. Hence, storage of set P via DAC satisfies the aim of unique decodability with constant time access in $\mathcal{C}$ in a space complexity of roughly $\Upsilon + \frac{\Upsilon}{b} + o(\frac{\Upsilon}{b})$ bits, where $\Upsilon = \sum_{\forall i}\lfloor\log p_i\rfloor = \sum_{i=1}^{n-1}\sum_{j=1}^{j\le i}|c_j|$. Note that instead of splitting an integer into fixed-length b–bits blocks, DAC runs an optimization to find the optimal variable length b values, and performs the split accordingly in practice.

It might be argued that instead of storing P, the storage of only the codeword lengths, which is actually the *gap* encoding of P, would take less space. However, in that case calculating the bit address of a codeword requires to sum all previous values. Since DAC does not support constant time prefix summation, one needs to store the absolute bit addresses.

Unlike DAC, an improved version of interpolative coding [14], namely the improved address-calculation (iAC) [12], supports prefix summation. Thus, in case of using iAC, it is

Original sequence T and original encoded sequence $\mathcal{C}(T)$:

a	e	b	f	d	c	b	b	d	h	b	b	g	f	a	a	a
0	10	1	11	01	00	1	1	01	001	1	1	000	11	0	0	0

Proposed rearrangement of $\mathcal{C}(T)$ via wavelet tree:

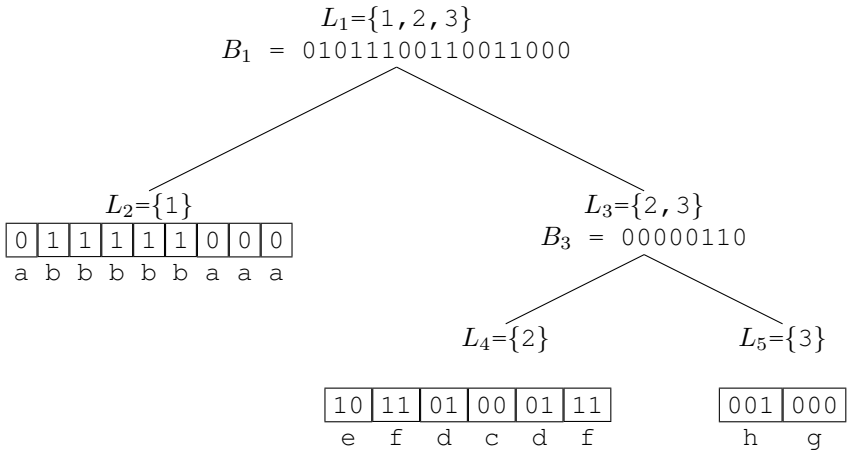

Fig. 1. Assuming the alphabet is $\Sigma = \{a, b, c, d, e, f, g, h\}$, and the corresponding codewords are $\mathcal{A} = \{0, 1, 00, 01, 10, 11, 000, 001\}$, the plain text $T = aebfdcbbdhbbgfaaa$ is encoded as $\mathcal{C}(T) = 0101110100110100111000011000$ with the non-prefix-free coding scheme $\mathcal{C} : \Sigma \rightarrow \mathcal{A}$. The proposed wavelet tree coding scheme applied on $\mathcal{C}(T)$ is shown.

possible to store only the codeword lengths in $\mathcal{C}(T)$, which might reduce the space consumption. The iAC coding scheme uses at most $n \cdot \log(\frac{|\mathcal{C}(T)|}{n}) + O(n)$ bits, and computes the sum of the first k numbers, for $1 \leq k \leq n$, in time $O(\log\log(n + |\mathcal{C}(T)|))$. When iAC coding is preferred, the access time becomes doubly-logarithmic instead of constant time. However, we noticed that the compressed prefix summation method defined in the study of Delpratt *et al.* (theorem 1 in [13]) provides constant time computation of the prefix sum in exactly the same space complexity of indexed AC-coding (but with a completely different way).

Table I summarizes the overhead space requirements of several alternatives to provide unique decodability of a non–prefix–free coding scheme, while at the same time supporting direct accessibility.

IV. Proposed Method

In this study we use wavelet trees to reorganize the encoded sequence $\mathcal{C}(T)$ such that the codewords with same lengths become consecutive, while conserving the ability of locating c_i to its exact position on $\mathcal{C}(T)$.

While creating an ordinary (binary) wavelet tree over an input sequence, the source alphabet is divided into two and a bitmap is generated by reserving one bit per symbol to mark which of the two sets the symbol at that position belongs to. Same procedure is recursively applied until there remains only one symbol at a leaf node.

Unlike the ordinary construction, instead of splitting the sequence according to the actual symbols, we split T according to the length of the corresponding codeword lengths. Remembering $L = \{\ell_1, \ell_2, \ldots, \ell_q\}$ is the set of distinct codeword lengths observed in codeword set $\mathcal{A}$, the depth of such a wavelet tree is $\log q$. Each leaf node of such a wavelet tree contains the juxtaposed codewords of a specific length.

Figure 1 depicts such a wavelet tree construction on a sample scenario, where the lengths of the codewords in $\mathcal{A}$ are 1, 2, or 3 as shown in the figure. Following the split of $L_1 = \{1, 2, 3\}$ into two as $L_2 = \{1\}$ and $L_3 = \{2, 3\}$, the root bitmap marks the positions of the symbols having codeword length 1 with 0, and those having lengths either 2 or 3 with 1. The left child of the root is a leaf node as there is a unique codeword length assigned to it, and contains the codewords of length 1 obeying their order of appearance in the original sequence. The right child of the root requires one more level of parsing, and the resulting leaf nodes are dedicated to codewords with lengths 2 and 3 respectively. The codewords at the leaf nodes are shown within frames.

Notice that the bitmaps B_1 and B_3 are the overheads of the porposed scheme, and need to be stored along with the actual code-streams at leaf nodes. In practice, the information regarding to the set L and the length of T is also required for proper decoding, which occupy negligible space.

The unique decodability provided by wavelet trees non-prefix-free coding schemes is stated in Theorem 1. Another feature of the proposed method is to bring direct accessibility to any variable-length coding scheme which is stated in Lemma 1.

Theorem 1: Any non-prefix-free coding scheme can be uniquely decoded by creating a wavelet tree over the encoded sequence. Such a wavelet tree occupies at most $n \cdot \lceil \log q \rceil$ bits,

where q is the number of distinct codeword lengths observed in the encoded sequence.

Proof: As the depth of the wavelet tree is $\lceil \log q \rceil$, and there might be at most n bits at each level, the space requirement is upper bounded with $n \cdot \lceil \log q \rceil$ bits. Notice that the length of the bitmap at each node can be calculated from its parent. The root bitmap has exactly n bits. The right and left child lengths are the total number of 0s and 1s in the parent node respectively. Whenever a leaf node is reached in left (or right) child, total number of bits used in that node to represent actual codewords is the number of 0s (or 1s) in the parent node times the codeword length dedicated to this leaf. Hence, when the $|T|$ and set L is known beforehand, nothing more is required for correct decoding. ■

Lemma 1: By reserving an additional $o(n \cdot \lceil \log q \rceil)$ bits in the proposed wavelet tree construction, any codeword in a sequence encoded with a variable–length coding scheme (including non–prefix–free codes), can be reached directly in at most $\lceil \log q \rceil$ steps regardless of the size of the sequence.

Proof: Assume we would like to reach the ith codeword by using the proposed wavelet tree. The leaf node specific to the length of the target codeword can be reached in at most $\lceil \log q \rceil$ node visits. At each visited node, we need to run a `rank` query. The `rank/select` queries over a binary string of length n can be performed in *constant* time by using an additional $o(n)$ bits via succinct data structures [15], [16], where there are also methods to run even in entropy compressed space of $n \cdot H_k + o(n)$ bits [17], [18]. Thus, the operation at each node takes constant time. When we arrive the leaf node, assuming i' is the latest result of the rank query performed on the parent node, and ℓ is the specific codeword length of the leaf, we move forward to the bit position $i' \cdot \ell + 1$ and read the next ℓ bits representing the ith symbol. ■

The pseudo-code of the procedure to access ith symbol of a sequence encoded with a variable–length scheme with the proposed wavelet tree structure is given in figure 2. As an example, let us follow the procedure to retrieve 4th codeword on the sample depicted in figure 1. We begin with the root as an internal node since $|L_1| > 1$. In the root bitmap B_1, we observe that the fourth bit is the second 1 bit. We branch to the right child, and move to the second bit. We observe that this is the second 0 bit on B_3. We go the left child, which is a leaf node dedicated to the code–length of 2 bits. The target codeword should be the second 2-bits block in that node, which is 11. That finishes our search as the codeword 11 encodes the 4th symbol (,which is `f`) of T.

ACCESS(i,PROPOSED WAVELET TREE STRUCTURE)

```
1.  curnode ← root
2.  a ← i
3.  while(1)
4.    B = b1b2... ← bitmap of the curnode
5.    L = {ℓ1, ℓ2, ...} ← the code-lengths of curnode
6.    if |L| = 1 //leaf node reached
7.      a ← a − 1
7.      return b_{ℓ1·a+1} b_{ℓ1·a+2} ... b_{ℓ1·a+ℓ1}
8.    else
9.      a ← rank(b_q, q, B)
10.     if (b_a = 0)
11.       curnode ← left child of curnode
12.     else
13.       curnode ← right child of curnode
```

Fig. 2. The pseudo-code of accessing ith codeword within the proposed wavelet tree encoding structure.

V. Experimental Results and Discussions

We compare different alternatives that are able to decode non-prefix-free codes with direct access capability.

The basic non-prefix-free coding of a given sequence would represent the most frequent two symbols with single bits 0 and 1. The next four symbols would be shown by the two–bits long codewords 00, 01, 10, and 11. Similarly, next 8, 16, ..., 2^r symbols ranked according to their frequencies will be shown by 3-bits, 4-bits, ..., r-bits long codewords.

Given sequence T and the corresponding alphabet $\Sigma = \{\epsilon_1, \epsilon_2, \ldots, \epsilon_\sigma\}$, the number of distinct codeword lengths would be at most $\lceil \log \frac{\sigma+2}{2} \rceil$ by solving the inequality $2 + 4 + 8 + \ldots + 2^r \leq \sigma$. Hence, for the non-prefix-free coding scheme $\mathcal{C} : \Sigma \rightarrow \mathcal{A}$ would require $\lceil \log \frac{\sigma+2}{2} \rceil$ codewords of lengths ranging from from 1 to r.

Certainly, the resulting bit-stream composed of juxtaposed codewords from $\mathcal{A} = \{0, 1, 00, 01, 10, 11, 000, \ldots\}$ is not uniquely decodable. As discussed previously we need to be able to compute the codeword boundaries for unique decodability, and hence, to store the beginning bit-positions of every codeword explicitly. This sampling should be made in a *dense* (for each codeword position) way to support direct access.

With this aim, the sequence of integers marking the codeword boundary positions can be encoded compactly via the dense sampling of Ferragina&Venturini [11] technique. A much better way is to use DAC of Brisaboa *et al.* [7], which outperforms FV according to the benchmarks given in the related paper.

A second alternative would be to store the individual codeword lengths instead of the boundary positions. Since lengths are much smaller values than positions, they are more likely to be compressed much better. However, in this case, one needs to efficiently compute the sum of the first $(i-1)$ code-lengths to retrieve the address of the ith codeword. By using the interpolative coding that can be done efficiently. We prefer to use the most recent state-of-the-art scheme by Elmasry *et al.* [12] during the experiments.

The third option is to rearrange the codewords via the wavelet tree AS proposed in this study, which conserves the ability to reconstruct original sequence.

In the setup of the benchmark, first, the initial 1 million bytes of the files *english*, *dblp.xml*, *DNA*, *protein*, and *sources* from Pizza&Chili corpus[1] are extracted and encoded with the basic non-prefix-free coding scheme as described above. Then, we create two integer sequences holding the codeword boundary bit positions and the codeword lengths. The position

[1] http://pizzachili.dcc.uchile.cl/index.html

			Space Usage bits/ symbol			Random Access μsec./ symbol		
file	σ	$\|\mathcal{C}(T)\|$	iAC	DAC	WT	iAC	DAC	WT
english	105	2503981	7.42	24.50	5.04	0.060	0.016	0.064
dblp	89	3005147	8.17	25.01	5.63	0.060	0.012	0.064
dna	6	1471215	5.61	22.47	2.52	0.056	0.008	0.028
protein	22	2523116	7.46	24.52	4.54	0.060	0.012	0.048
sources	98	3113442	8.33	25.11	5.83	0.060	0.016	0.072

TABLE II
COMPARISON OF ALTERNATIVE SOLUTIONS PROVIDING UNIQUE DECODABILITY WITH DIRECT ACCESS CAPABILITY.

sequence is represented via the DAC [7], and the length sequence is coded with the improved AC-coding (iAC) [12], which are the best solutions to our knowledge.

Table II lists the comparison of the three alternatives. [2] The alphabet size is shown by σ. Total length of the code–stream generated by encoding the one million symbols via the non-prefix-free coding scheme in bits is given in $|\mathcal{C}(T)|$ column. The space usage is measured by bits per symbol, and is computed by $\frac{|\mathcal{C}(T)|+overhead}{|T|}$.

The overhead in iAC is the length of the compact representation of codeword lengths supporting prefix summation, and the overhead in DAC is the total length of the compressed codeword starting bit positions. In case of the proposed method, that overhead is the size of the wavelet tree supporting direct access to the rearranged codewords. Constant time rank queries both in DAC and wavelet schemes require additional space, and we consider 5 percent increase in both cases by wasting an integer (32–bits) in each of every 640 bits to store cumulative number of 1s for rank queries. Time required to access a random symbol is measured by taking the average of one million random access in all cases. The machine we used during the benchmark is a 64-bit Intel i7 processor with 16 GB memory running Ubuntu 12.04.

As observed in table II, the space usage is worst in DAC, which is expected since the bit positions array include numbers much higher than those exist in the codeword lengths sequence. Space consumption is significantly reduced in the proposed scheme, being the leader in all cases, especially the DNA case. Wavelet tree coding provides 30 percent gain in space, where it reaches 50 percent in case of DNA sequence.

In terms of time, DAC is the most speedy solution in accessing a randomly selected item, where iAC and wavelet are compatible. Notice that current implementation of wavelet has not been optimized yet, and we expect a better speed in a better engineered optimized version.

VI. CONCLUSIONS

We have shown a new application area of wavelet trees in variable-length coding by introducing a method to provide unique decodability of non-prefix-free codes, which also supports direct access. When compared to dense sampling techniques [11], implemented with most recent advances ([7], [12]), that can be considered in solving the target problem, the wavelet tree based coding reports best space usage. Optimizations, both in engineering and algorithmic enhancements, will be sought to improve speed without any sacrifice of space.

Despite saving space, we believe that having an elegant solution for the non-prefix-free codes might open new avenues of research in data compression and related areas. Notice that creating a wavelet tree over the code-lengths will provide direct access in any variable-length coding scheme with a price of $n\cdot\log q+o(n\cdot\log q)$. Design and analysis of the wavelet tree adopted implementations of variable-length coding schemes, e.g., Elias, Golomb, Rice [5], might be considered in future studies.

[2]The author thanks S.Ladra and J.Teuhola for sharing their implementations of DAC and iAC.

REFERENCES

[1] R. Grossi, A. Gupta, and J. S. Vitter, "High-order entropy-compressed text indexes," in *Proceedings of the 14th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2003, pp. 841–850.
[2] G. Navarro, "Wavelet trees for all," in *CPM 2012: Combinatorial Pattern Matching*, ser. LNCS, 2012, vol. 7354, pp. 2–26.
[3] T. Gagie, G. Navarro, and S. J. Puglisi, "New algorithms on wavelet trees and applications to information retrieval," *Theoretical Computer Science*, vol. 426-427, pp. 25–41, 2012.
[4] P. Ferragina, R. Giancarlo, and G. Manzini, "The myriad virtues of wavelet trees," *Information and Computation*, vol. 207, no. 8, pp. 849–866, 2009.
[5] P. Fenwick, *Lossless Compression Handbook*. Academic Press, 2003, ch. 3, pp. 55–78.
[6] L. G. Kraft, "A device for quantizing, grouping, and coding amplitude modulated pulses," Master's thesis, Dept. of Electrical Engineering, MIT, Cambridge, Mass., 1949.
[7] N. R. Brisaboa, S. Ladra, and G. Navarro, "DACs: Bringing direct access to variable length codes," *Information Processing and Management*, vol. 49, no. 1, 2013.
[8] H. E. Williams and J. Zobel, "Compressing integers for fast file access," *The Computer Journal*, vol. 42, pp. 193–201, 1999.
[9] M. Dalai and R. Leonardi, "Non prefix-free codes for constrained sequences," in *Proceedings of International Symposium on Information Theory (ISIT)*, 2005, pp. 1534 –1538.
[10] *Compressing integer sequences and sets*. Springer, 2008, ch. Encyclopedia of algorithms, pp. 178–182.
[11] P. Ferragina and R. Venturini, "A simple storage scheme for strings achieving entropy bounds," *Theoretical Computer Science*, vol. 372, pp. 115–121, 2007.
[12] A. Elmasry, J. Katajainen, and J. Teuhola, "Improved address–calculation coding of integer arrays," in *SPIRE 2012: String Processing and Information Retrieval*, ser. LNCS, 2012, vol. 7608, pp. 205–216.
[13] O. Delpratt, N. Rahman, and R. Rahman, "Compressed prefix sums," in *(SOFSEM 2007: Theory and Practice of Computer Science*, ser. LNCS, 2007, vol. 4362, pp. 235–247.
[14] J. Teuhola, "Interpolative coding of integer sequences supporting log–time random access," *Information Processing and Management*, vol. 47, 2011.
[15] G. Jacobson, "Space-efficient static trees and graphs," in *Proceedings of the 30th Annual Symposium on Foundations of Computer Science (FOCS)*, 1989, pp. 549–554.
[16] D. R. Clark and J. I. Munro, "Efficient suffix trees on secondary storage," in *Proceedings of the 7th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 1996, pp. 383–391.
[17] R. Raman, V. Raman, and S. S. Rao, "Succinct indexable dictionaries with applications to encoding k–ary trees and multisets," in *Proceedings of the 13th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2002, pp. 233–242.
[18] M. Patrascu, "Succincter," in *Proceedings of 49th IEEE Symposium on Foundations of Computer Science (FOCS)*. IEEE Computer Society, 2008, pp. 305–313.