LETTER *Special Section on Information Theory and Its Applications*

Direct- or Fast-Access Decoding Schemes for VF Codes

Hirosuke YAMAMOTO[†a)], ***Fellow and*** **Yuka KUWAORI**[†*b)], ***Nonmember***

SUMMARY In this paper, we propose two schemes, which enable any VF code to realize direct- or fast-access decoding for any long source sequence. Direct-access decoding means that any source symbol of any position can be directly decoded within constant time, not depending on the length of source sequence N, without decoding the whole codeword sequence. We also evaluate the memory size necessary to realize direct-access decoding or fast-access decoding with decoding delay $O(\log\log N)$, $O(\log N)$, and so on, in the proposed schemes.

key words: *VF code, direct-access decoding, rank function, select function*

1. Introduction

Fixed-to-variable length (FV) codes like Huffman codes and variable-to-fixed length (VF) codes like Tunstall codes are often used to store big data efficiently. But, since the FV codes have variable codeword length and the VF codes have variable parse length, we must decode the codeword sequence from the beginning even if we want to decode only one source symbol of a long source sequence $x_1x_2\cdots x_N$. In the case of big data with very large N, the decoding delay $O(N)$ is not acceptable for the decoding of only one or a few source symbols.

In order to overcome this defect, direct-access decoding schemes have been studied such that any x_j can be decoded within constant time.

In the case of FV codes, several direct- or fast-access decoding schemes [1]–[7] have been proposed, which uses the so-called wavelet tree [1], [2], rank and/or select functions. Especially, in the case of Huffman codes, the direct-access decoding can be realized with the same coding rate asymptotically as the ordinary Huffman code if we use the same shape of wavelet tree as the Huffman code tree [2], [4].

For a binary sequence $\boldsymbol{b} = b_1b_2\cdots b_n$, rank function $\mathrm{rank}(\boldsymbol{b}, l)$ and select function $\mathrm{select}(\boldsymbol{b}, \ell)$ are defined as follows.

$$\mathrm{rank}(\boldsymbol{b}, l) = \sum_{\ell=1}^{l} b_\ell, \tag{1}$$

$$\mathrm{select}(\boldsymbol{b}, \ell) = \min\{l : \mathrm{rank}(\boldsymbol{b}, l) = \ell\}. \tag{2}$$

So, the rank function $\mathrm{rank}(\boldsymbol{b}, l)$ gives the number of "1" included in the first l bits of $\boldsymbol{b}$, and the select function $\mathrm{select}(\boldsymbol{b}, \ell)$ gives the position of the ℓ-th "1" in $\boldsymbol{b}$. The rank and select functions can be calculated with constant time and $n + o(n)$ memory space when the length of $\boldsymbol{b}$ is n [8]–[10].

On the other hand, few direct-access decoding schemes have been studied in the case of VF codes. Yoshida-Sasakawa-Sekine-Kida (YSSK) [11] proposed a direct-access decoding scheme such that bit b_j is assigned to each x_j of source sequence $\boldsymbol{x} = x_1x_2\cdots x_N$, and b_j is set as $b_j = 1$ if x_j is the last source symbol included in the same codeword, and $b_j = 0$ otherwise. YSSK scheme can be applied to any VF code. But, since YSSK scheme requires one bit b_j for each x_j, YSSK scheme is inefficient when the size of source alphabet is small. Especially, if the source alphabet is binary, YSSK scheme cannot attain any compression.

In this paper, in order to improve the above defect of YSSK scheme, we propose a modified YSSK scheme with a devised data structure such that after we divide both a source sequence and its codeword sequence into blocks, we apply YSSK scheme to the sequence of blocks. Although the modified YSSK scheme can attain good performance even for binary source sequences, the performance depends on how to implement the rank and select functions. So, by combining the idea of the modified YSSK scheme with the data structure proposed by Kimura-Suzuki-Sugano-Koike [7] to realize efficiently the rank function, we also proposed a self-contained scheme, which does not use rank and select functions, to realize the direct-access decoding or fast-access decoding with decoding delay $O(\log\log N)$, $O(\log N)$, and so on, for VF codes.

The modified YSSK scheme and the latter scheme are treated in Sects. 2 and 3, respectively.

The following notation is used in this paper.

Notation

- $\boldsymbol{x}$: a source sequence $\boldsymbol{x} = x_1x_2\cdots x_N$.
- x_j : the j-th symbol of $\boldsymbol{x}$.
- $\boldsymbol{y}$: the sequence of codewords $\boldsymbol{y} = y_1y_2\cdots y_n$, which is encoded from $\boldsymbol{x}$ by a VF code.
- y_i: the i-th codeword of $\boldsymbol{y}$. All y_i have the same length because of VF coding.
- N: the length of $\boldsymbol{x}$.
- n: the length of $\boldsymbol{y}$, i.e. the number of codewords included in $\boldsymbol{y}$.
- L_i: the number of source symbols encoded into codeword y_i.

Manuscript received January 31, 2016.

[†]The authors are with the Department of Complexity Science and Engineering, The University of Tokyo, Kashiwa-shi, 277-8561 Japan.

[*]Presently, with NS Solutions Corporation.

a) E-mail: hirosuke@ieee.org

b) E-mail: t.yuka.116@gmail.com

DOI: 10.1587/transfun.E99.A.2291

$\overline{L}$: the average of L_i, i.e., $\overline{L} \equiv N/n$.
L^+: $L^+ \equiv \max_i L_i$.
L^-: $L^- \equiv \min_i L_i$.
Γ_i: the number of source symbols included in the first i codewords $y_1 y_2 \cdots y_i$, i.e., $\Gamma_i \equiv \sum_{l=1}^{i} L_l$, $\Gamma_n = N$.

2. Modified YSSK Scheme

Assume that a source sequence $\boldsymbol{x}$ is encoded to the sequence of codewords $\boldsymbol{y}$ by a VF code. In YSSK scheme, one bit b_j is assigned to each x_j as follows[†].

$$b_j = \begin{cases} 1, & \text{if } x_j \text{ is the first symbol of a codeword.} \\ 0, & \text{otherwise.} \end{cases} \tag{3}$$

For any j, we can decode x_j directly by using $\boldsymbol{b} = b_1 b_2 \cdots b_N$ as follows. Let $i = \mathrm{rank}(\boldsymbol{b}, j)$ and $m = j - \mathrm{select}(\boldsymbol{b}, i) + 1$. Then, the x_j is the m-th source symbol included in codeword y_i. Note that y_i can easily be obtain from $\boldsymbol{y}$ since every codeword has the same length in VF codes. In YSSK scheme, we require N bits to store $\boldsymbol{b}$ besides $\boldsymbol{y}$. So, if x_j is binary, the total memory size becomes larger than the size of $\boldsymbol{x}$.

In order to overcome the above defect, we divide $\boldsymbol{x}$ into blocks with length A as $\boldsymbol{x} = \hat{\boldsymbol{x}}_1 \hat{\boldsymbol{x}}_2 \cdots \hat{\boldsymbol{x}}_{N/A}$[††], where $\hat{\boldsymbol{x}}_u$ is the u-th source block defined by $\hat{\boldsymbol{x}}_u = x_{A(u-1)+1} \cdots x_{Au}$. We also divide $\boldsymbol{y}$ into blocks with length B as $\boldsymbol{y} = \hat{\boldsymbol{y}}_1 \hat{\boldsymbol{y}}_2 \cdots \hat{\boldsymbol{y}}_{n/B}$, where $\hat{\boldsymbol{y}}_v$ is the v-th codeword block defined by $\hat{\boldsymbol{y}}_v = y_{B(v-1)+1} \cdots y_{Bv}$. Note that we can easily obtain $\hat{\boldsymbol{y}}_v$ from $\boldsymbol{y}$ because each y_i has the same length in the case of VF codes. Each $\hat{\boldsymbol{y}}_v$ includes at least BL^- source symbols. So, in order to guarantee that each $\hat{\boldsymbol{x}}_u$ are encoded within two consecutive $\hat{\boldsymbol{y}}_v$ and $\hat{\boldsymbol{y}}_{v+1}$, we assume that A and B satisfy $A < BL^-$.

Then, we assign one bit $b_u \in \{0, 1\}$ to each source block $\hat{\boldsymbol{x}}_u$ as follows.

(a) $b_1 = 1$.
(b) For $u \geq 2$, $b_u = 1$ if the first source symbol of $\hat{\boldsymbol{x}}_u$ is included in a codeword block $\hat{\boldsymbol{y}}_v$, but the first source symbol of $\hat{\boldsymbol{x}}_{u-1}$ is not included in the same $\hat{\boldsymbol{y}}_v$. Otherwise, $b_u = 0$.

Note that each $\hat{\boldsymbol{y}}_v$ corresponds to only one $\hat{\boldsymbol{x}}_u$ with $b_u = 1$, and hence the number of u with $b_u = 1$ is equal to the number of v, i.e., n/B. But, the beginning of $\hat{\boldsymbol{x}}_u$ with $b_u = 1$ does not coincide with the beginning of $\hat{\boldsymbol{y}}_v$ generally. So, we record the difference in source symbol indexes for each v. Let $f_{\mathrm{b}}(v)$ represent the index of the first source symbol included in $\hat{\boldsymbol{y}}_v$. Then, the difference d_v is given by $d_v = A(u-1) + 1 - f_{\mathrm{b}}(v)$ if "$b_u = 1$" is the v-th "1" in $\boldsymbol{b}$, and d_v takes a value in $\{0, 1, \cdots, A-1\}$.

[†]Although the *last symbol* is used to set $b_j = 1$ in [11], we use the *first symbol* of a codeword for simplicity.

[††]For simplicity, we assume that N and n can be divided by A and B, respectively. If not so, the last $\hat{\boldsymbol{x}}_{\lceil N/A \rceil}$ and $\hat{\boldsymbol{y}}_{\lceil n/B \rceil}$ have shorter lengths than the others.

In more detail, b_u and d_v can be obtained from $\boldsymbol{x} = \hat{\boldsymbol{x}}_1 \hat{\boldsymbol{x}}_2 \cdots \hat{\boldsymbol{x}}_{N/A}$ and $\boldsymbol{y} = \hat{\boldsymbol{y}}_1 \hat{\boldsymbol{y}}_2 \cdots \hat{\boldsymbol{y}}_{n/B}$ by the following algorithm, where $f_{\mathrm{e}}(v)$ represents the index of the last source symbol included in $\hat{\boldsymbol{y}}_v$. Note that $f_{\mathrm{b}}(v)$ and $f_{\mathrm{e}}(v)$ can easily be obtained when $\boldsymbol{x}$ is encoded into $\boldsymbol{y}$ sequentially.

Algorithm 1 (Encoding):
Step1 Encode $\boldsymbol{x}$ into $\boldsymbol{y} = \hat{\boldsymbol{y}}_1 \hat{\boldsymbol{y}}_2 \cdots \hat{\boldsymbol{y}}_{n/B}$ by a VF code, and obtain $f_{\mathrm{b}}(v)$ and $f_{\mathrm{e}}(v)$.
Step2 Set $b_1 = 1$, $u = 2$, $v = 1$.
Step3 If $f_{\mathrm{b}}(v) \leq A(u-1) + 1 \leq f_{\mathrm{e}}(v)$,
$b_u = 0$,
else (i.e., if $f_{\mathrm{e}}(v) < A(u-1) + 1$),
$b_u = 1$, $v = v + 1$, $d_v = A(u-1) + 1 - f_{\mathrm{b}}(v)$.
Step4 If $u = N/A$, exit,
else, $u = u + 1$, go to Step3.

Note that the indexes j of the first and last source symbols included in $\hat{\boldsymbol{y}}_v$ are given by $A \times [\mathrm{select}(\boldsymbol{b}, v) - 1] - d_v + 1$ and $A \times [\mathrm{select}(\boldsymbol{b}, v+1) - 1] - d_{v+1}$, respectively. Hence, x_j can be decoded directly from $\boldsymbol{y}$, $\boldsymbol{b} = b_1 b_2 \cdots b_{n/B}$ and $\boldsymbol{d} = d_1 d_2 \cdots d_{n/B}$ as follows.

Algorithm 2 (Direct-access decoding):
Step1 $v = \mathrm{rank}\left(\boldsymbol{b}, \lceil \frac{j}{A} \rceil\right)$.
Step2 If $j > A \times [\mathrm{select}(\boldsymbol{b}, v+1) - 1] - d_{v+1}$,
then $v = v + 1$.
Step3 $m_{\mathrm{b}} = j - A \times [\mathrm{select}(\boldsymbol{b}, v) - 1] + d_v$,
$m_{\mathrm{e}} = A \times [\mathrm{select}(\boldsymbol{b}, v+1) - 1] - d_{v+1} - (j-1)$.
Step4 x_j is the m_{b}-th source symbol obtained by decoding $\hat{\boldsymbol{y}}_v$ from the beginning, and x_j is also the m_{e}-th source symbol obtained by decoding $\hat{\boldsymbol{y}}_v$ backward from the end.

As an example, assume that each y_i has L_i shown in Fig. 1. Then d_v and b_u are obtained by using Algorithm 1 as shown in Figs. 1 and 2, respectively. Furthermore, for instance, x_{76} can be directly decoded as follows.

Example 1: Assume that $A = 5$ and $B = 4$ are used, and L_i, d_v and b_u are given as shown in Figs. 1 and 2.

Step1 For $j = 76$ and $A = 5$, $v = \mathrm{rank}(\boldsymbol{b}, \lceil \frac{76}{5} \rceil) = \mathrm{rank}(\boldsymbol{b}, 16) = 7$.
Step2 x_{76} is included in $\hat{\boldsymbol{y}}_7$ since it holds that
$76 \leq 5 \times [\mathrm{select}(\boldsymbol{b}, 8) - 1] - d_8 = 5[18 - 1] - 4 = 81$.
Step3

$$\begin{aligned} m_{\mathrm{b}} &= j - A \times [\mathrm{select}(\boldsymbol{b}, v) - 1] + d_v \\ &= 76 - 5 \times [\mathrm{select}(\boldsymbol{b}, 7) - 1] + d_7 \\ &= 76 - 5 \times [15 - 1] + 1 \\ &= 7 \\ m_{\mathrm{e}} &= A \times [\mathrm{select}(\boldsymbol{b}, v+1) - 1] - d_{v+1} - (j-1) \\ &= 5 \times [\mathrm{select}(\boldsymbol{b}, 8) - 1] - d_8 - 75 \\ &= 5 \times [18 - 1] - 4 - 75 \\ &= 6 \end{aligned}$$

Step4 x_{76} is the 7-th source symbol obtained by decoding $\hat{\boldsymbol{y}}_v$ forward from the beginning, and x_{76} is also the 6-th

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
L_i	4	3	2	4	4	2	3	4	4	3	3	2	3	4	2	2	3	3	2	2	4	2	2	2	4	3	2	3	2	4	2	3
Γ_i	4	7	9	13	17	19	22	26	30	33	36	38	41	45	47	49	52	55	57	59	63	65	67	69	73	76	78	81	83	87	89	92
v	1				2				3				4				5				6				7				8			
d_v	0				2				4				2				1				1				1				4			

Fig. 1 An example of L_i, Γ_i, d_v for $A = 5$ and $B = 4$.

u	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
b_u	1	0	0	1	0	0	1	0	1	0	1	0	1	0	1	0	0	1	0

Fig. 2 b_u in the case of Fig. 1.

source symbol by decoding $\hat{\boldsymbol{y}}_v$ backward from the end.

In Step 4 of Algorithm 1, we need to decode at most $B/2$ codewords y_i to obtain x_j. But, decoding time does not depends on N.

Next we consider the necessary memory size to store $\boldsymbol{b}$ and $\boldsymbol{d}$. The length of $\boldsymbol{b}$ is N/A, and d_v satisfies $0 \le d_v \le A-1$ and $d_1 = 0$. Hence, the total memory size M_{mYSSK} is given by

$$\begin{aligned} M_{\mathrm{mYSSK}} =& \frac{N}{A} + \left(\frac{n}{B} - 1\right)\lceil \log A \rceil \\ <& \frac{N}{A} + \frac{N(\log A + 1)}{B\overline{L}} \\ <& \frac{N(\log A + 2)}{A}. \end{aligned} \tag{4}$$

because $N = \overline{L}n$ and $A < BL^- \le B\overline{L}$. Hence, by setting A a little large and setting B as $A < BL^-$, we can decrease the memory size considerably compared with the original YSSK scheme, which requires N bits.

It is worth noting that both YSSK scheme and the modified YSSK scheme require another memory space to store the data structure to calculate rank and select functions of $\boldsymbol{b}$ within constant time. Hence, the performance of these schemes depends on how to implement these functions.

3. Self-Contained Scheme

An efficient data structure to calculate rank function is proposed in [7]. So, by combining the data structures used in Sect. 2 and [7], we construct a self-contained scheme that requires neither the rank function nor the select function in this section.

In the modified YSSK scheme, $\boldsymbol{y}$ is divided into blocks $\hat{\boldsymbol{y}}_v$ with fixed length B independently from $\hat{\boldsymbol{x}}_u$. But, in this section, we divide $\boldsymbol{y}$ into blocks with variable length such that each block $\hat{\boldsymbol{y}}_v$ almost corresponds to each $\hat{\boldsymbol{x}}_v$, which has fixed length A, where $v = 1, 2, \cdots, \frac{N}{A}$. Then, we define $\hat{\boldsymbol{y}}_v$ such that the index i_v of the last codeword included in $\hat{\boldsymbol{y}}_v$ is given by

$$i_v = \max\{i : \Gamma_i \le A \cdot v\}. \tag{5}$$

Now we define $d_v^{(1)}$, which is the difference between $A \cdot v$ and Γ_{i_v}, as follows.

$$d_v^{(1)} \equiv A \cdot v - \Gamma_{i_v}. \tag{6}$$

Table 1 Memory Size of i_v and $d_v^{(1)}$.

	number	bits
i_v	$\frac{N}{A}$	$\lceil \log n \rceil$
$d_v^{(1)}$	$\frac{N}{A}$	$\lceil \log L^+ \rceil$

Then, for direct-access decoding we use $\{i_v\}, \{d_v^{(1)}\}$, $v = 1, 2, \cdots, N/A$, which can easily be obtained in the encoding of $\boldsymbol{x}$. For simplicity, we set $d_0^{(1)} = 0$.

For any j, x_j can be directly decoded from $\boldsymbol{y}$, $\{i_v\}$, and $\{d_v^{(1)}\}$ by the following algorithm.

Algorithm 3 (Direct access decoding):
Step1 $v = \lceil \frac{j}{A} \rceil$
Step2 If $j > A \cdot v - d_v^{(1)}$, then $v = v + 1$.
Step3 $m_{\mathrm{b}} = j - [A \cdot (v-1) - d_{v-1}^{(1)}]$,
$m_{\mathrm{e}} = [A \cdot v - d_v^{(1)}] - j + 1$.
Step4 x_j is the m_{b}-th source symbol obtained by decoding $\boldsymbol{y}$ from $y_{i_{v-1}+1}$, and x_j is also the m_{e}-th source symbol obtained by decoding $\boldsymbol{y}$ backward from y_{i_v}.

In order to obtain x_j, we need to decode at most $A/2$ codewords. On the other hand, the memory sizes to store $\{i_v\}$ and $\{d_v^{(1)}\}$ are shown in Table 1. Hence the total memory size is given by

$$\frac{N}{A}\left\lceil \log \frac{N}{\overline{L}} \right\rceil + \frac{N}{A}\lceil \log L^+ \rceil < \frac{N}{A}\left(\log \frac{N}{\overline{L}} + \log L^+ + 2\right) \tag{7}$$

where $N = n\overline{L}$.

Although the above scheme is self-contained, it is not efficient because Eq. (7) is larger than Eq. (4). Therefore, we improve the performance by using a data structure similar to [7], i.e., we further divide each block $\hat{\boldsymbol{y}}_v$ into sub-blocks $\hat{\boldsymbol{y}}_{v,w}$ with variable length such that each sub-block $\hat{\boldsymbol{y}}_{v,w}$ almost corresponds of $\hat{\boldsymbol{x}}_{v,w}$, which is the sub-block of $\hat{\boldsymbol{x}}_v = \hat{\boldsymbol{x}}_{v,1}\hat{\boldsymbol{x}}_{v,2}\cdots\hat{\boldsymbol{x}}_{v,A/C}$ and each $\hat{\boldsymbol{x}}_{v,w}$ has fixed length C.[†]

Then, if $w = A/C$, the index of the last codeword included in $\hat{\boldsymbol{y}}_{v,w}$ is given by i_v. For $1 \le w \le A/C - 1$, we define $\hat{\boldsymbol{y}}_{v,w}$ such that the index of the last codeword included in $\hat{\boldsymbol{y}}_{v,w}$ is given by $i_{v-1} + i_{v,w}$ where $i_{v,w}$ is defined by

$$\begin{aligned} i_{v,w} \equiv \max\{l : & \Gamma_{i_{v-1}+l} \le A \cdot (v-1) + C \cdot w, \\ & l < i_v - i_{v-1}\}. \end{aligned} \tag{8}$$

We also define $d_{v,w}^{(2)}$ by

[†]For simplicity, we assume that A can be divided by C.

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
L_i	4	3	2	4	4	2	3	4	4	3	3	2	3	4	2	2	3	3	2	2	4	2	2	2	4	3	2	3	2	4	2	3
Γ_i	4	7	9	13	17	19	22	26	30	33	36	38	41	45	47	49	52	55	57	59	63	65	67	69	73	76	78	81	83	87	89	92
v									1												2											3
i_v									9												21											32
$d_v^{(1)}$									2												1											4
w		1		2			3		4			1			2			3			4			1			2			3		4
$i_{v,w}$		2		4			7					3			6			9						3			6			9		
$d_{v,w}^{(2)}$		1		3			2		2			2			1			1			1			3			2			1		4

Fig. 3 An example of L_i, Γ_i, v, i_v, $d_v^{(1)}$, w, $i_{v,w}$, $d_{v,w}^{(2)}$ for $A = 32$ and $C = 8$.

$$d_{v,w}^{(2)} \equiv A \cdot (v-1) + C \cdot w - \Gamma_{i_{v-1}+i_{w,v}}. \tag{9}$$

Note that Eqs. (8) and (9) correspond to (5) and (6) in the one-stage division, respectively.

Then, for direct decoding, we use $\{i_v\}$, $\{i_{w,v}\}$, and $\{d_{v,w}^{(2)}\}$, which can easily be obtained in the encoding of $\boldsymbol{x}$. For simplicity, we set $d_{v,0}^{(2)} = 0$, and also note that $d_v^{(1)} = d_{v,\frac{A}{C}}^{(2)}$.

Let i_b and i_e represent the indexes of the first and last codewords included in $\hat{\boldsymbol{y}}_{v,w}$, respectively, and let j_b and j_e represent the indexes of the first source symbol included in y_{i_b} and the last source symbol included in y_{i_e}, respectively. Then, x_j can be directly decoded from $\boldsymbol{y}$, $\{i_v\}$, $\{i_{w,v}\}$, and $\{d_{v,w}^{(2)}\}$ as follows.

Algorithm 4 (Direct access decoding):

Step1 $v = \lceil \frac{j}{A} \rceil$.

Step2 If $j > A \cdot v - d_v^{(1)}$,
then $v = v + 1$, $w = 1$, go to Step 5.

Step3 $w = \left\lceil \frac{j - A\cdot(v-1)}{C} \right\rceil$

Step4 If $j > A \cdot (v-1) + C \cdot w - d_{v,w}^{(2)}$, then $w = w + 1$.

Step5

$$\begin{aligned}
i_\mathrm{b} &= i_{v-1} + i_{v,w-1} + 1,\\
j_\mathrm{b} &= A \cdot (v-1) + C \cdot (w-1) - d_{v,w-1}^{(2)} + 1,\\
m_\mathrm{b} &= j - j_\mathrm{b} + 1,\\
i_\mathrm{e} &= i_{v-1} + i_{v,w},\\
j_\mathrm{e} &= A \cdot (v-1) + C \cdot w - d_{v,w}^{(2)},\\
m_\mathrm{e} &= j_\mathrm{e} - j + 1.
\end{aligned}$$

Step6 x_j is the m_b-th source symbol obtained by decoding $\boldsymbol{y}$ from y_{i_b}, and x_j is also the m_e-th source symbol obtained by decoding $\boldsymbol{y}$ backward from y_{i_e}.

As an example, consider the same L_i as Fig. 1. Then, for $A = 32$, $C = 8$, $\{i_v\}$, $\{i_{w,v}\}$, and $\{d_{v,w}^{(2)}\}$ are obtained as shown in Fig. 3. Furthermore, for instance, x_{79} can be directly decoded as follows.

Example 2: Assume that $A = 32$ and $C = 8$ are used, and $\{i_v\}$, $\{i_{w,v}\}$, and $\{d_{v,w}^{(2)}\}$ are given as shown in Fig. 3.

Step1 $v = \lceil \frac{79}{32} \rceil = 3$

Step2 x_{79} is included in $\hat{\boldsymbol{y}}_3$ since it holds that $79 \le 32 \times 3 - d_3^{(1)} = 96 - 4 = 92$.

Step3 $w = \left\lceil \frac{79 - 32\times(3-1)}{8} \right\rceil = 2$

Step4 Since it holds that $79 > 32 \times (3-1) + 8 \times 2 - d_{3,2}^{(2)} = 64 + 16 - 2 = 78$, set $w = 2 + 1 = 3$. This means that x_{79} is included in $\hat{\boldsymbol{y}}_{3,3}$.

Step5

$$\begin{aligned}
i_\mathrm{b} &= i_{3-1} + i_{3,3-1} + 1 = 21 + 6 + 1 = 28,\\
j_\mathrm{b} &= 32 \times (3-1) + 8 \times (3-1) - d_{3,3-1}^{(2)} + 1\\
&= 64 + 16 - 2 + 1 = 79,\\
m_\mathrm{b} &= 79 - 79 + 1 = 1,\\
i_\mathrm{e} &= i_{3-1} + i_{3,3} = 21 + 9 = 30,\\
j_\mathrm{e} &= 32 \times (3-1) + 8 \times 3 - d_{3,3}^{(2)} = 64 + 24 - 1\\
&= 87,\\
m_\mathrm{e} &= 87 - 79 + 1 = 9.
\end{aligned}$$

Step6 x_{79} is the first source symbol obtained by decoding $\boldsymbol{y}$ from y_{28}, and x_{79} is also the 9-th source symbol obtained by decoding $\boldsymbol{y}$ backward from y_{30}.

Table 2 Memory size of i_v, $i_{v,w}$, and $d_{v,w}^{(2)}$.

	number	bits
i_v	$\frac{N}{A}$	$\lceil \log n \rceil$
$i_{v,w}$	$\frac{N}{C} - \frac{N}{A}$	$\lceil \log \frac{A}{L^-} \rceil$
$d_{v,w}^{(2)}$	$\frac{N}{C}$	$\lceil \log L^+ \rceil$

In this scheme, we must decode at most $C/2$ codewords to obtain x_j for any j, and we need the memory size shown in Table 2 to store $\{i_v\}$, $\{i_{w,v}\}$, and $\{d_{v,w}^{(2)}\}$. Hence, from $N = n\overline{L}$, the total memory size M is given by

$$\begin{aligned}
M &= \frac{N}{A}\left(\left\lceil \log \frac{N}{\overline{L}} \right\rceil - \left\lceil \log \frac{A}{L^-} \right\rceil\right)\\
&\qquad + \frac{N}{C}\left(\left\lceil \log \frac{A}{L^-} \right\rceil + \lceil \log L^+ \rceil\right) \qquad (10)\\
&< \frac{N}{A}\left(\log \frac{N}{\overline{L}} - \log \frac{A}{L^-} + 1\right)\\
&\qquad + \frac{N}{C}\left(\log \frac{A}{L^-} + \log L^+ + 2\right). \qquad (11)
\end{aligned}$$

Note that A can be selected independently of the decoding delay, which depends on C in the two-stage scheme. Hence, by using relatively large A, the total memory size can be decreased considerably compared with (7).

Next we consider the order of M. From (10), we have that

$$M = O\left(\frac{N}{A} \log \frac{N}{A} + \frac{N}{C} \log A\right). \tag{12}$$

Table 3 The order of total memory size.

A	C	M
$\frac{\log N}{\log\log N}$	constant	$O(N\log\log N)$
$\log N$	$\log\log N$	$O(N)$
$(\log N)^{1+a}$	$(\log N)^a$	$O\left(\frac{N\log\log N}{(\log N)^a}\right)$
N^b	bN^b	$O(N^{1-b}\log N)$

Hence, in the case of the direct-access decoding with C = constant, the oder of M becomes

$$M = O(N\log\log N) \tag{13}$$

if we use $A = O((\log N)/(\log\log N))$. In the case of the fast-access decoding with $C = \log\log N$, $C = (\log N)^a$, or $C = bN^b$ for $a > 0$ and $1 \gg b > 0$, the order of M can be further reduced to $O(N)$, $O((N\log\log N)/(\log N)^a)$, $O(N^{1-b}\log N)$, respectively, as shown in Table 3.

4. Conclusion

In this paper, we proposed two schemes that enable the direct-access decoding for any VF codes. The first scheme is a modified scheme of [11], which uses rank and select functions. The second scheme is a self-contained scheme, which does not require rank and select functions.

References

[1] R. Grossi, A. Gupta, and J.S. Vitter, "High-order entropy-compressed text indexes," Proc. 14th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), pp.841–850, 2003.

[2] G. Navarro, "Wavelet trees for all," Combinatorial Pattern Matching, Lecture Notes in Computer Science, vol.7354, pp.2–26, Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[3] N.R. Brisaboa, A. Fariña, S. Ladra, and G. Navarro, "Reorganizing compressed text," Proc. 31st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR'08, pp.139–145, 2008.

[4] N.R. Brisaboa, S. Ladra, and G. Navarro, "DACs: Bringing direct access to variable-length codes," Inform. Process. Manage., vol.49, no.1, pp.392–404, 2013.

[5] M.O. Külekci, "Uniquely decodable and directly accessible non-prefix-free codes via wavelet trees," Proc. 2013 IEEE International Symposium on Information Theory, pp.1969–1973, 2013.

[6] M.O. Külekci, "Enhanced variable-length codes: Improved compression with efficient random access," Proc. 2014 Data Compression Conference, pp.362–371, 2014.

[7] K. Kimura, Y. Suzuki, S. Sugano, and A. Koike, "Computation of rank and select functions on hierarchical binary string and its application to genome mapping problems for short-read DNA sequences," J. Comput. Biol., vol.16, no.11, pp.1601–1613, 2009.

[8] G. Jacobson, "Space-efficient static trees and graphs," Proc. 30th Annual Symposium on Foundations of Computer Science, pp.549–554, 1989.

[9] G. Jacobson, Succinct Static Data Structure, PhD Thesis, Carnegie Mellon University, 1989.

[10] D. Clark, Compact Pat Tree, PhD Thesis, University of Waterloo, 1996.

[11] S. Yoshida, H. Sasakawa, K. Sekine, and T. Kida, "Direct access to variable-to-fixed length codes with a succinct index," Proc. 2014 Data Compression Conference, p.436, 2014.