ISIT'2013 1569717361

Almost Instantaneous FV Codes

Hirosuke Yamamoto
Dep. of Complexity Science & Engineering
The University of Tokyo
Kashiwa-shi, Chiba, Japan
Email: hirosuke@ieee.org

Xiaofeng Wei
Dep. of Complexity Science & Engineering
The University of Tokyo
Kashiwa-shi, Chiba, Japan
Email: wei@it.k.u-tokyo.ac.jp

***Abstract*—In this paper, K-ary almost instantaneous fixed-to-variable-length (AIFV) codes are proposed for $K \geq 3$, and it is shown that the K-ary AIFV codes using $K-1$ code trees can attain better compression than K-ary Huffman codes for stationary memoryless sources. Furthermore, it is also shown that binary relaxed AIFV codes with two code trees can beat binary Huffman codes.**

I. Introduction

Lossless source codes are classified into fixed-to-variable-length (FV) codes and variable-to-fixed-length (VF) codes, which can be represented by code trees and parsing trees, respectively. It is well known that Huffman codes and Tunstall codes can attain the best compression in FV codes and VF codes, respectively, for stationary memoryless sources if a fixed code tree or a fixed parsing tree are used. But, Yamamoto-Yokoo [1] proposed the almost instantaneous VF codes (AIVF codes), which can attain better compression than Tunstall codes. An AIVF code uses $|\mathcal{X}|-1$ parsing trees for a source alphabet $\mathcal{X}$ and codewords are assigned to incomplete internal nodes in addition to leaves in each parsing tree. Although instantaneous encoding is not possible since incomplete internal nodes are used for encoding, the AIVF code is devised so that the encoding delay is at most one source symbol, and hence the code is called *almost instantaneous*. Furthermore, Yoshida-Kida [2][3] showed that any AIVF code can be encoded and decoded by a single integrated parsing tree and the total number of nodes can be considerably reduced by the integration.

In the case of FV codes, any uniquely decodable FV code must satisfy Kraft's inequality, and such a code can be realized by an instantaneous FV code, i.e., a prefix FV code. Hence, the Huffman code which is the best code in the class of instantaneous FV codes is also the best code in the class of uniquely decodable FV codes. However, it is assumed implicitly in the above argument that only a single fixed code tree is used. Hence, we can devise more efficient codes than Huffman codes if multiple code trees can be used in the same way as the AIVF codes.

In section II, we propose K-ary almost instantaneous FV codes (AIFV codes) for stationary memoryless sources for $K \geq 3$. A K-ary AIFV code uses $K-1$ code trees, and source symbols are assigned to incomplete internal nodes in addition to leaves. We also show how an AIFV code can be encoded and decoded. In section III, we derive an inequality corresponding to Kraft's inequality, and in section IV we give a greedy algorithm to obtain good AIFV codes, which can beat Huffman codes. In the AIFV codes, decoding is not instantaneous, but the decoding delay is at most one code symbol. In this case, the AIFV codes can be constructed only for $K \geq 3$. However, in section V, we show that if decoding delay is relaxed to at most two code symbols, we can devise binary AIFV codes that can beat binary Huffman codes.

II. Definition of AIFV codes

Let X be a stationary memoryless source over a finite alphabet $\mathcal{X}$ with probability $P(x)$ for $x \in \mathcal{X}$. A codeword is a K-ary sequence, each symbol of which takes a value of $\mathcal{Y} = \{1, 2, \cdots, K\}$. Then, an FV code is called an AIFV code if it satisfies the following conditions.

Definition 1 (Code trees of AIFV code):

(A) The AIFV code consists of $K-1$ code trees, $T_0, T_1, \cdots, T_{K-2}$.

(B) For $0 \leq k \leq K-2$, the root of T_k has $K-k$ children, and the j-th branch connecting the root to the j-th child has code symbol $k+j$ for $1 \leq j \leq K-k$.

(C) If a node is a complete internal node, i.e., it has K children, then, j-th branch connecting the node to the j-th child has code symbol j for $1 \leq j \leq K$.

(D) If a node is an incomplete internal node, then the number of children of the node, say m, must be less than or equal to $K-2$, and j-th branch connecting the incomplete node to the j-th child has code symbol j for $1 \leq j \leq m$.

(E) In every code tree T_k, each source symbol $x \in \mathcal{X}$ is assigned to a leaf or an incomplete internal node. Any source symbol must not be assigned to any complete internal node. The codeword of x is the sequence of code symbols over the path from the root to the corresponding leaf or incomplete internal node.

Example 1: A simple example of an AIFV code is shown in Fig. 1, where $\mathcal{X} = \{a, b, c, d, e\}$, $\mathcal{Y} = \{1, 2, 3\}$, and $K = 3$. When T_0 is used, the codewords of a, b, c, d, e are 1, 2, 3, 21, 31, respectively. But, they are 2, 21, 31, 32, 33, respectively, when T_1 is used.

For code tree T_k, let $\mathcal{N}_0^{(k)}$ and $\mathcal{N}_m^{(k)}$ be the sets of leaves and internal nodes with m children, respectively, and let n_x be the node or leaf corresponding to a source symbol x. Then, a source sequence $\boldsymbol{x} = x_1 x_2 \cdots x_N$ is encoded by using code trees $T_0, T_1, \cdots, T_{K-2}$ as follows.

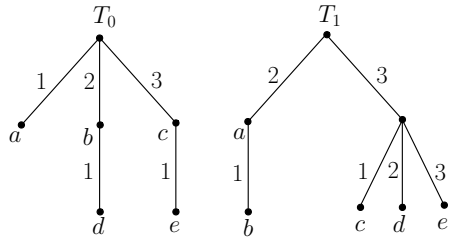

Fig. 1. An AIFV code for $K = 3$.

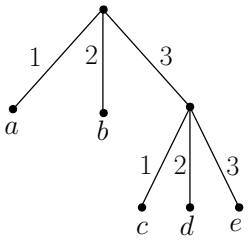

Fig. 2. Huffman code tree.

Procedure 1 (Encoding of AIFV code):

(a) Let $k = 0$.

(b) Repeat (b-1) and (b-2) for $s = 1, 2, \cdots, N$.

(b-1) Encode x_s by code tree T_k.

(b-2) If $n_{x_s} \in \mathcal{N}_m^{(k)}$, then let $k = m$. (Note that $0 \le m \le K-2$ from the property (D) of AIFV code trees.)

For the AIFV code given by Fig. 1, source sequence "$abac$" is encoded to codeword sequence "1 2 2 31", and source sequence "$cdebac$" is encoded to codeword sequence "3 32 31 2 2 31".

Procedure 2 (Decoding of AIFV code):

(a) Let $k = 0$.

(b) Repeat (b-1) and (b-2) until the codeword sequence becomes null.

(b-1) By following the codeword sequence, trace a path from the root in T_k as long as possible. If we reach a leaf, then output the source symbol assigned to the leaf. Let $k = 0$. If we reach an incomplete internal node included in $\mathcal{N}_m^{(k)}$ and the next code symbol is not included in $\{1, 2, \cdots, m\}$, then output the source symbol assigned to the incomplete internal node. Let $k = m$.

(b-2) Remove the decoded codeword from the codeword sequence.

Remark 1: Since some source symbols are assigned to incomplete internal nodes, the AIFV code is not an instantaneous code. But, when x_i is encoded by an incomplete internal node, we can decode x_i by checking only one code symbol next to the codeword of x_i. Hence, the codewords of AIFV codes can be decoded with the delay of at most one code symbol.

Remark 2: In Procedures 1 and 2, we assumed that the end of codeword sequence can be detected by another mechanism. In the case that the end cannot be detected, we add a special symbol EOF to $\mathcal{X}$, and we assign EOF to a leaf in each T_k. By encoding EOF at the end of $\boldsymbol{x}$, we can know the end in the decoding. This worsens the compression ratio a little. But, the degradation is negligible if N is large.

Now we show that an AIFV code can attain better compression than the corresponding Huffman code. Consider the source and codeword alphabets $\mathcal{X}$ and $\mathcal{Y}$ treated in Example 1, and assume that $P(x) = 1/5$ for all $x \in \mathcal{X}$. Then, the entropy of this source is $H(X) = \log_3 5 \approx 1.465$. The Huffman code of this source is given by Fig. 2, and the average codeword length of the Huffman code L_H is given by $L_H = 1.6$.

Let L_k be the average codeword length of T_k, $0 \le k \le K-2$. Then, from Fig.1, L_0 and L_1 are given by $L_0 = 1.4$ and $L_1 = 1.8$, respectively. We note from Step (b-2) in Procedure 1 that the state transition probability of code trees $Q(T_m|T_k)$ is given by

$$Q(T_m|T_k) = \sum_{x:n_x \in \mathcal{N}_m^{(k)}} P(x). \tag{1}$$

In the case of the AIFV code trees shown in Fig.1, we have $Q(T_1|T_0) = 0.4$ and $Q(T_0|T_1) = 0.8$, and $Q(T_0) = 2/3$ and $Q(T_1) = 1/3$, where $Q(T_0)$ and $Q(T_1)$ are the stationary probability of T_0 and T_1, respectively. Finally, we obtain the average codeword length L_{AIFV} as follows.

$$L_{AIFV} = \frac{2}{3}L_0 + \frac{1}{3}L_1 = \frac{4.6}{3} \approx 1.533, \tag{2}$$

which is shorter than $L_H = 1.6$.

Now we explain the reason why the AIFV code can beat the Huffman code. Since incomplete internal nodes are used for encoding in T_0, L_0 can become smaller than $H(X)$. L_1 is larger than L_H because the root of T_1 has only two children. But, from $Q(T_0) > Q(T_1)$, L_{AIFV} becomes smaller than L_H in the above example.

Remark 3: It is well known that the coding rate of Huffman codes can be improved by extending alphabet $\mathcal{X}$ to $\mathcal{X}^2$. But, the size of a Huffman code tree must also be squared. If we use an AIFV code, the total size of AIFV code trees is about $K-1$ times the size of an FV code for $\mathcal{X}$. Therefore, an AIFV code can attain a good performance with low complexity compared with the alphabet extension especially if $|\mathcal{X}|$ is large. As shown in Section IV, AIFV codes can often beat even Huffman codes with alphabet extension $\mathcal{X}^2$. Furthermore, it is also possible to apply AIFV coding to extended alphabets $\mathcal{X}^2, \mathcal{X}^3, \cdots$.

III. Bounds of average codeword length

In this section, we derive lower and upper bounds of average codeword length L_k for code tree T_k, $0 \le k \le K-2$. We first consider T_0, and let $l_0(x)$ be the codeword length of $x \in \mathcal{X}$ in T_0. If $n_x \in \mathcal{N}_m^{(0)}$, then we can change the node n_x into an complete internal node by adding $K-m$ children, the depth of which is $l_0(x)+1$. Hence, we have from Kraft's inequality that

$$\sum_{m=0}^{K-2} \sum_{x:n_x \in \mathcal{N}_m^{(0)}} (K-m) K^{-[l_0(x)+1]} \le 1, \tag{3}$$

where the equality holds if all leaves and incomplete internal nodes are used to assign source symbols.

In the case of $k > 0$, since the root of T_k has only $K - k$ children, $K^{-[l_0(x)+1]}$ should become $(K-k)^{-1}K^{-l_k(x)}$. Therefore, we have

$$\sum_{m=0}^{K-2} \sum_{x:n_x \in \mathcal{N}_m^{(k)}} (K-m)(K-k)^{-1}K^{-l_k(x)} \le 1. \tag{4}$$

Let $\hat{P}(x) = (K-m)(K-k)^{-1}K^{-l_k(x)}$ if $n_x \in \mathcal{N}_m^{(k)}$. Then, from $\sum_{x\in\mathcal{X}} \hat{P}(x) = 1$ and $-\log_K \hat{P}(x) = l_k(x) + \log_K(K-k) - \log_K(K-m)$, we have

$$\begin{aligned} 0 \le D(P\|\hat{P}) &= \sum_{x\in\mathcal{X}} P(x)\log_K \frac{P(x)}{\hat{P}(x)} \\ &= -H(X) + \left[\sum_{x\in\mathcal{X}} P(x)l_k(x)\right] + \log_K(K-k) \\ &\quad - \sum_{m=0}^{K-2} \sum_{x:n_x\in\mathcal{N}_m^{(k)}} P(x)\log_K(K-m) \\ &= -H(X) + L_k + \log_K(K-k) \\ &\quad - \sum_{m=0}^{K-2} P(\mathcal{N}_m^{(k)})\log_K(K-m), \end{aligned} \tag{5}$$

where $P(\mathcal{N}_m^{(k)}) = \sum_{x:n_x\in\mathcal{N}_m^{(k)}} P(x)$. Hence, L_k must satisfy that

$$L_k \ge H(X) + \sum_{m=0}^{K-2} P(\mathcal{N}_m^{(k)})\log_K \frac{K-m}{K-k} \tag{6}$$

Next we derive an upper bound of L_k. We can easy show that an AIFV code tree T_k can be realized if (4) is satisfied. We now define $l_k(x)$ as $l_k(x) = \lceil -\log_K P(x) + \log_K(K-m)/(K-k)\rceil < -\log_K P(x) + \log_K(K-m)/(K-k) + 1$. Then, $l_k(x)$ satisfies (4), and hence, there exists T_k satisfying that

$$\begin{aligned} L_k &= \sum_{x\in\mathcal{X}} P(x)l_k(x) \\ &< H(X) + \sum_{m=0}^{K-2} P(\mathcal{N}_m^{(k)})\log_K \frac{K-m}{K-k} + 1. \end{aligned} \tag{7}$$

The term $\log_K(K-m)/(K-k)$ in (6) and (7) is negative (resp. positive) if $m > k$ (resp. $m < k$). Therefore, L_k becomes small (resp. large) when k is small (resp. large).

The total average codeword length L_{AIFV} is obtained by

$$L_{AIFV} = \sum_k Q(T_k)L_k, \tag{8}$$

where $Q(T_k)$ is the stationary probability of T_k, and $Q(T_k)$ is determined from $Q(T_m|T_k) = P(\mathcal{N}_m^{(k)})$, $0 \le k \le K-2$, $0 \le m \le K-2$. Unfortunately, the bound of L_{AIFV} derived from (6), (7), (8) becomes $H(X) \le L_{AIFV} < H(X)+1$, which is the same as the well-known bound of instantaneous FV codes. But, since AIFV code trees have larger degree of freedom than instantaneous FV code trees, we can construct AIFV codes achieving better compression than Huffman codes.

IV. Generation of AIFV code trees

Since it is generally difficult to determine the optimal AIFV code trees for a given source probability $P(x)$, we give a greedy algorithm, Algorithm 1, to obtain good AIFV code trees. To describe the algorithm we use the following notation. $\mathcal{N}^{(k)}$: the set of all leaves and all incomplete internal nodes in T_k, i.e., $\mathcal{N}^{(k)} = \cup_{m=0}^{K-2}\mathcal{N}_m^{(k)}$, m_n: the number of children of node $n \in \mathcal{N}^{(k)}$ ($m_n = 0$ if n is a leaf), x_n: source symbol assigned to n. For simplicity, we assume without loss of generality that $\mathcal{X} = \{x_1, x_2, \cdots, x_{|\mathcal{X}|}\}$ is arranged in the order of probability, i.e. $P(x_i) \ge P(x_{i+1})$ for all i, and $|\mathcal{X}| \ge K$.

In Algorithm 1, we grow a code tree T_k one node by one node from the root to leaves, as opposed to the Huffman code, which is constructed from leaves to the root. We assign a source symbol x_i to a created node in the order of i, i.e. in the order of $P(x_i)$.

In Step 1 of Algorithm 1, we create the initial tree with the root and its $K-k$ children.

Assume that some part of T_k has already been constructed. Consider the case that we make a new child from $n \in \mathcal{N}^{(k)}$ satisfying $0 \le m_n \le K-3$ and assign a source symbol $\hat{x}$ to the new child. In this case, $\hat{x}$ has codeword length $l_k(x_n)+1$ and the contribution to L_k is given by

$$P(\hat{x})(l_k(x_n)+1). \tag{9}$$

If we consider only T_k, it is desirable from the second term of (6) that the number of children of an incomplete internal node is increased. However, when x_n is encoded by T_k, the code tree state changes from T_k to T_{m_n}, and L_m becomes larger as m increases. Therefore, we must consider this cost to minimize L_{AIFV}.

Consider the ideal case that every sibling node occurs with equal probability in all T_k. In this case, since T_{m_n} has only $K-m_n$ children at the root, the average codeword length of T_{m_n} is longer than the one of T_0, and the difference is given by $1-\log_K(K-m_n)$. Hence, in the ideal case, the cost caused by creating one child at n is evaluated by

$$\begin{aligned} &P(x_n)(\{1-\log_K(K-m_n-1)\} - \{1-\log_K(K-m_n)\}) \\ &\quad = P(x_n)\log_K \frac{K-m_n}{K-m_n-1}. \end{aligned} \tag{10}$$

For simplicity, we also apply this cost function to nonideal cases.

If $m_n = K-2$, i.e., n has $K-2$ children, we must create two new children from n to make n a complete internal node, and we assign $\hat{x}$ to one of the new children and change the assignment of x_n from n to the other new child. In this case, the ideal cost caused by changing the assignment of x_n is given by $P(x_n)(1-\{1-\log_K 2\}) = P(x_n)\log_K 2$, which is equal to (10) when $m_n = K-2$. Therefore, (10) can be used for any m_n, $0 \le m_n \le K-2$.

Combining (9) and (10), the total cost can be measured by

$$C(n|\hat{x}) = P(\hat{x})(l_k(x_n)+1) + P(x_n)\log_K \frac{K-m_n}{K-m_n-1} \tag{11}$$

when a source symbol $\hat{x}$ is assigned to the new child created from n. In Step 4 of Algorithm 1, a new child is created from the node $\hat{n}$ that minimizes the above cost function.

To minimize L_k for a fixed shape of T_k, a source symbol x with larger probability must have a shorter codeword or larger m_{n_x} if the codeword length is equal. Hence, the following condition must be satisfied.

C1 $l_k(x_i) \le l_k(x_{i+1})$
or
$[l_k(x_i) = l_k(x_{i+1})$ and $m_{n_{x_i}} \le m_{n_{x_{i+1}}}]$

In Step 5.a of Algorithm 1, the assignment of x_i and x_{i+1} is swapped if C1 is not satisfied.

Let $\mathcal{N}_{all}^{(k,d)}$ be the set of all leaves and all complete and incomplete internal nodes with depth d in T_k, and let $\mathcal{N}_{all}^{(k)} = \bigcup_d \mathcal{N}_{all}^{(k,d)}$. Then, for $n \in \mathcal{N}_{all}^{(k)}$, we define the weight of n by

$$W(n) = \sum_{x:n_x=n \text{ or descendant of } n} P(x). \tag{12}$$

Then, to minimize L_k for the case that m_n is fixed for all $n \in \mathcal{N}_{all}^{(k)}$, n with larger weight must have shorter depth. Hence, the following condition must be satisfied.

C2 For $n \in \mathcal{N}_{all}^{(k,d)}$ and $\tilde{n} \in \mathcal{N}_{all}^{(k,d+1)}$, $W(n) \ge W(\tilde{n})$

In Step 5.b of Algorithm 1, the subtree of n and $\tilde{n}$ are swapped if C2 is not satisfied.

Algorithm 1 (Generation of T_k)

1. Make the root and $K-k$ children of the root. Assign x_i, $1 \le i \le K-k$ to each child. Let $\mathcal{N}^{(k)} = \{$the children of the root$\}$, and let $\tilde{\mathcal{X}} = \{x_1, x_2, \cdots, x_{K-k}\}$ and $\mathcal{X} = \mathcal{X} \setminus \tilde{\mathcal{X}}$.
2. If $\mathcal{X}$ is null, exit.
3. Let $\hat{x}$ be the first element of $\mathcal{X}$, which has the largest probability in $\mathcal{X}$, and let $\tilde{\mathcal{X}} = \tilde{\mathcal{X}} \cup \{\hat{x}\}$ and $\mathcal{X} = \mathcal{X} \setminus \{\hat{x}\}$.
4. Let $\hat{n} = \operatorname{argmin}_{n \in \mathcal{N}^{(k)}} C(n|\hat{x})$.
   - If $0 \le m_{\hat{n}} \le K-3$, then create one child from $\hat{n}$ and assign $\hat{x}$ to the child. Let $\mathcal{N}^{(k)} = \mathcal{N}^{(k)} \cup \{$created child$\}$.
   - If $m_{\hat{n}} = K-2$, then create two children from $\hat{n}$ and assign $\hat{x}$ and $x_{\hat{n}}$ to two created children. Let $\mathcal{N}^{(k)} = \mathcal{N}^{(k)} \setminus \{\hat{n}\} \cup \{$created two children$\}$.
5. a. Check whether every i, $1 \le i \le |\tilde{\mathcal{X}}| - 1$, satisfies Condition C1. If there exists i that does not satisfy C1, swap the assignment of x_i and x_{i+1}. Repeat this process until all i satisfy C1.
   b. Check whether every depth d satisfies Condition C2. If there exist $n \in \mathcal{N}_{all}^{(k,d)}$ and $\tilde{n} \in \mathcal{N}_{all}^{(k,d+1)}$ that do not satisfy C2, swap the subtrees of n and $\tilde{n}$. Repeat this process until all d satisfy C2.
   c. If T_k is not modified in Steps 5.a and 5.b, then go to Step 2. Otherwise go to Step 5.a.

V. Numerical results

We consider AIFV codes constructed by applying Algorithm 1 to the following three types of sources for numerical evaluation of compression performance.

$$P_0(x_i) = \frac{1}{|\mathcal{X}|}, \tag{13}$$

$$P_1(x_i) = A_1(|\mathcal{X}| + 1 - i), \tag{14}$$

$$P_2(x_i) = A_2(|\mathcal{X}| + 1 - i)^2, \tag{15}$$

where A_1 and A_2 are normalization constants.

The cost function $f(K,m) \equiv \log_K \frac{K-m}{K-m-1}$ used in (11) is valid theoretically only in the ideal case. But, by applying Algorithm 1 to the above sources, we numerically confirmed that when f, $0 < f < 1$, is used instead of $f(K,m)$ in (11), L_{AIFV} is actually minimized at $f = f(K,m)$.

Now we compare the compression performance of AIFV codes obtained by Algorithm 1 with Huffman codes and Huffman codes with the alphabet extension $\mathcal{X}^2$.

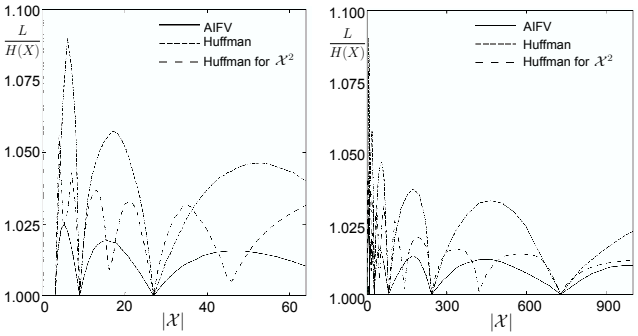

Fig. 3. Comparison of AIFV code with Huffman code for $K = 3$ and P_0.

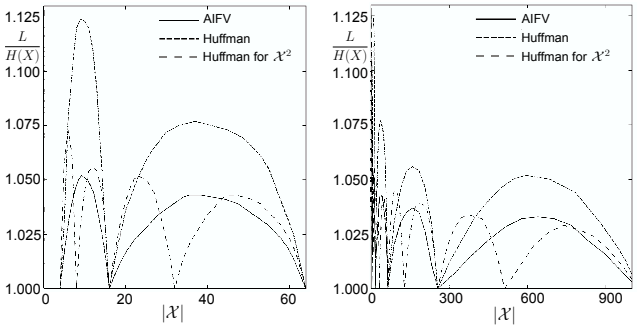

Fig. 4. Comparison of AIFV code with Huffman code for $K = 4$ and P_0.

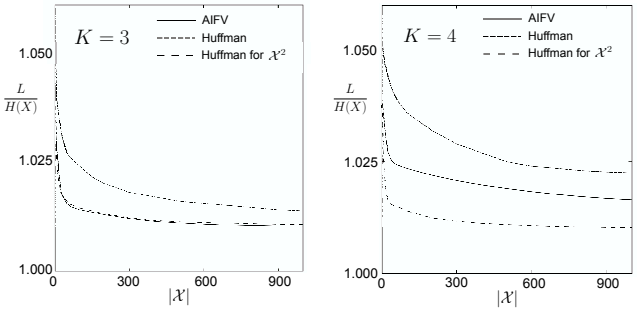

Fig. 5. Comparison of AIFV code with Huffman code for P_1.

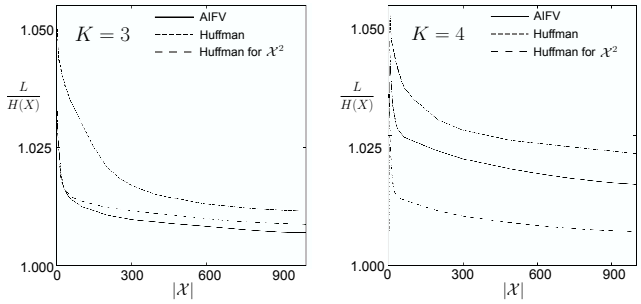

Fig. 6. Comparison of AIFV code with Huffman code for P_2.

For the source P_0, Figs. 3 and 4 show the performance for the cases of $K = 3$ and $K = 4$, respectively, and Figs.5 and 6 treat P_1 and P_2, respectively. The vertical axis is the average codeword length per source symbol normalized by the source entropy, and the horizontal axis is alphabet size $|\mathcal{X}|$. We note from these figures that AIFV codes beat Huffman codes in any cases. For P_0, AIFV codes can often achieve better performance than Huffman codes with the alphabet extension $\mathcal{X}^2$ in both of $K = 3$ and $K = 4$. Furthermore, for P_1 and P_2, AIFV codes can attain almost the same performance as Huffman codes with $\mathcal{X}^2$ for $K = 3$ although Huffman codes with $\mathcal{X}^2$ have better performance than AIFV codes for $K = 4$.

VI. Binary Relaxed AIFV Codes

The AIFV codes defined in section II can be constructed only for $K \geq 3$, and K-ary AIFV codes with binary representation of code symbols are not so good as binary Huffman codes. But, by relaxing the decoding delay to at most two code symbols, i.e., 2 bits, we can construct binary AIFV codes that attain better compression than binary Huffman codes.

We show an example of a binary relaxed AIFV code in Fig. 7. As shown in the figure, a binary relaxed AIFV code consists of two code trees T_0 and T_1. The root of T_0 has two children and can have four grandchildren. But, the root of T_1 must have just three grandchildren connected by paths 01, 10 and 11. On the other hand, every incomplete internal node with a source symbol must have only one grandchild connected by path 00. Note that no source symbol is assigned to the internal nodes denoted by squares in Fig. 3, and hence, the codeword corresponding to an internal node can be decoded by checking whether the next two bits are 00 or not.

For $\mathcal{X} = \{a, b, c, d\}$ with $P(a) = 0.45$, $P(b) = 0.3$, $P(c) = 0.2$, and $P(d) = 0.05$, the binary relaxed AIFV code given in Fig. 7 has the average codeword length $L_0 = 1.65$ and $L_1 = 2.1$ for T_0 and T_1, respectively. Since T_1 is used only just after c is encoded in this example, we have $Q(T_1|T_0) = 0.2$ and $Q(T_0|T_1) = 0.8$ which mean that $Q(T_0) = 0.8$ and $Q(T_1) = 0.2$. Therefore, we have $L_{AIFV} = 1.65 \times 0.8 + 2.1 \times 0.2 = 1.74$, which is better than the average codeword length of the binary Huffman code $L_H = 1.8$.

In the same way as Section IV, we can devise a cost function and an algorithm similar to Algorithm 1 to obtain good binary relaxed AIFV codes. We show only the numerical results for

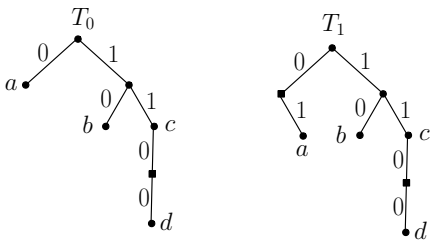

Fig. 7. A binary relaxed AIFV code.

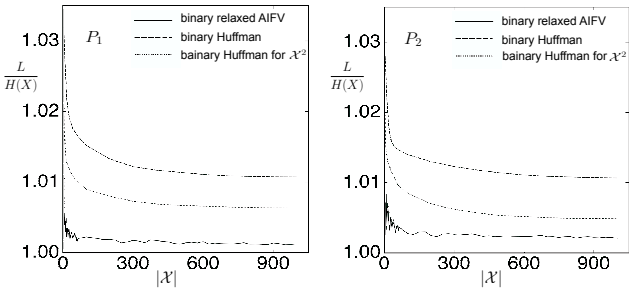

Fig. 8. Comparison of binary AIFV codes with binary Huffman codes

P_1 and P_2 in Fig. 8 because the binary relaxed AIFV codes have the same performance with the Huffman codes for P_0. From these figures, the binary relaxed AIFV codes can attain better performance than even the binary Huffman codes with the alphabet extension $\mathcal{X}^2$ for P_1 and P_2.

VII. Concluding Remarks

We proposed K-ary AIFV codes for $K \geq 3$ and binary relaxed AIFV codes. The decoding delays of AIFV codes and relaxed AIFV codes are at most one and two code symbols, respectively. We showed that the proposed AIFV codes can attain better compression than Huffman codes for three kinds of sources, P_0, P_1, and P_2. We note that Fabris [4] considered a variable-to-variable-length code (VV code) constructed by concatenating a Huffman code to a Tunstall code. It might be possible to construct a good VV code by concatenating a binary relaxed AIFV code to an AIVF code.

Acknowledgment

This work was supported in part by JSPS Grant-in-Aid for Scientific Research (Challenging Exploratory Research) No.2465240.

References

[1] H.Yamamoto and H.Yokoo, "Average-sense optimality and competitive optimality for almost instantaneous VF codes," IEEE Trans. on Inform. Theory, vol.47, no.6, pp.2174-2184, Sep. 2001

[2] S.Yoshida and T.Kida, "An efficient algorithm for almost instantaneous VF code using multiplexed parse trees," DCC 2010, pp.219-228, 2010

[3] S.Yoshida and T.Kida, "Analysis of multiplexed parse trees for almost instantaneous VF codes," 2012 IIAI International Conference on Advanced Applied Informatics (IIAIAAI 2012), pp.36-41, 2012

[4] F.Fabris, "Variable-length -to- variable-length source coding: a greedy step-by-step algorithm", IEEE Trans. on Inform. Theory, vol.38, no.5, pp.1609-1617, Sep. 1992