IEICE TRANS. INF. & SYST., VOL.E93–D, NO.12 DECEMBER 2010

PAPER

On-Line Electrocardiogram Lossless Compression Using Antidictionary Codes for a Finite Alphabet

Takahiro OTA[†a)], ***Member and*** **Hiroyoshi MORITA**[††b)], ***Senior Member***

SUMMARY An antidictionary is particularly useful for data compression, and on-line electrocardiogram (ECG) lossless compression algorithms using antidictionaries have been proposed. They work in real-time with constant memory and give better compression ratios than traditional lossless data compression algorithms, while they only deal with ECG data on a binary alphabet. This paper proposes on-line ECG lossless compression for a given data on a finite alphabet. The proposed algorithm gives not only better compression ratios than those algorithms but also uses less computational space than they do. Moreover, the proposed algorithm work in real-time. Its effectiveness is demonstrated by simulation results.

key words: *antidictionary, electrocardiogram, lossless, real-time, data compression*

1. Introduction

In recent years, information and communication technologies have been used in the biomedical field. In medical communication technology, various applications have been proposed [1]–[3]. Moreover, there have been regulations and standards for wireless medical devices, networks, and applications since secure, high-quality, high-reliability links, and the ability to work with the other medical systems are required [4]. On the other hand, in medical information technology, one of this field research, called biomedical informatics, deals with the resources, devices and methods to optimize acquisition, storage, retrieval and use of biomedical information. It becomes more important to use electronic biomedical data in digital format to be handled by computer systems and networks. Data compression is particularly useful in their systems where resources are scarce, e.g., limited bandwidth in communication systems and the capacity of storage systems.

ElectroCardioGram (ECG) is one of biomedical data. ECG compression algorithms are required for two main reasons, that is, an effective and economic data storage and reducing communication costs. Moreover, ECG wearable measurement devices require to conserve energy and storage to extend the life of the battery and the storage for as long as possible [5]. Data compression algorithms are also useful for the requirements. On the other hand, each ECG wave has an almost same waveform while a wave differs from other waves with respect to the period and the amplitude. Moreover, a few arrhythmias occur. These properties make it difficult to compress ECG by means of lossless data compression, so that numerous lossy data compression algorithms of ECG have been proposed [6], [7]. However, from the point of view of clinical medicine, an on-line lossless compression algorithm is needed since ECG compression algorithms are required to retrieve the original ECG signals and to work in real-time [8].

Lossless data compression algorithms typically use a dictionary, which is the set of all substrings of an input data, to construct statistical models and replace substrings with indices in the dictionary (e.g., [9], [10]). On the other hand, in 1999, Crochemore *et al.* proposed an off-line lossless data compression algorithm using an antidictionary of an input binary string [11]. An antidictionary for a given string is the set of all words of minimal length, called Minimal Forbidden Words (MFWs), which never appear in the string. Their method, called Data Compression using Antidictionary (DCA), was applied to the Calgary Corpus, that is a well-known benchmark archives for data compression, and it was shown to perform as well as popular compression algorithms such as the Lempel-Ziv (LZ) algorithms [13]. It was also proved that the DCA algorithm achieves a compression rate for a balanced binary source that is equal to its entropy rate. Some algorithms based on the DCA also have been proposed [14]–[16]. Those algorithms work in an off-line manner, that is, a static scheme. In 2006, an online linear compression algorithm based on the DCA algorithm was proposed [17]. This algorithm, called Arithmetic Coding based on the DCA (ACDCA), gives 20% improvement in average compressed file size relative to the DCA algorithm [12]. Moreover, some algorithms based on the ACDCA also have been proposed [18], [19].

Regarding ECG lossless compression, in 2004, an online lossless data compression algorithm using antidictionaries of a given binary ECG data was proposed [20]. Experimental results showed that this algorithm achieves better compression ratios than those of the original DCA and the LZ algorithms for files on the MIT-BIH arrhythmia database [21]. In 2009, an arithmetic coding based on the algorithm was proposed [22]. These algorithms work in real-time with constant computational memory since they use an antidictionary constructed from training data of constant length of an input ECG data.

Manuscript received February 15, 2010.
Manuscript revised August 12, 2010.

[†]The author is with the Dept. of Electronic Engineering, Nagano Prefectural Institute of Technology, Ueda-shi, 386–1211 Japan.
[††]The author is with the Graduate School of Information Systems, University of Electro-Communications, Chofu-shi, 182–8585 Japan.
a) E-mail: ota@pit-nagano.ac.jp
b) E-mail: morita@is.uec.ac.jp

DOI: 10.1587/transinf.E93.D.3384

However, these algorithms can deal with only binary ECG data since compression ratios get worse as alphabet size increases since the size of an antidictionary is proportional to alphabet size [15]. To deal with ECG data over a finite alphabet, we can apply the ACDCA to ECG data, however, it is difficult to handle an extremely long data such as ECG since the ACDCA requires computational memory in proportional to the data size.

In this paper, we propose a new on-line ECG lossless compression based on the ACDCA with aim to handle ECG data over a finite alphabet and improve compression ratios. The proposed algorithm works in real-time and uses small computational memory in coder size relative to the traditional algorithm [22]. Their effectivenesses are demonstrated by simulation results.

This paper is organized as follows. Section 2 gives a review of DCA algorithms and contains the basic notations and definitions. Section 3 gives a proposed algorithm, and Sect. 4 evaluates the proposed algorithm by computer simulations for files of the MIT-BIH Arrhythmia Database. Section 5 summarizes our results.

2. Review of ECG Lossless Compression Using Antidictionaries

2.1 Basic Definitions and Notations

Let $\mathcal{X}$ be a finite source alphabet $\{\xi_0, \xi_1, \ldots, \xi_{n-1}\}$ and $\mathcal{X}^*$ be the set of all finite strings over $\mathcal{X}$, including the null string of length zero, denoted by λ. For a string $\boldsymbol{x} = x_1 x_2 \ldots x_n \in \mathcal{X}^*$, let $\boldsymbol{x}^i$ be a *prefix* of $\boldsymbol{x}$ of length i, that is,

$$\boldsymbol{x}^i = \begin{cases} x_1 x_2 \ldots x_i & (1 \le i \le n), \\ \lambda & (i = 0). \end{cases} \tag{1}$$

Let $\mathcal{S}(\boldsymbol{x})$ be the set of all *suffixes* of $\boldsymbol{x}$, that is,

$$\mathcal{S}(\boldsymbol{x}) = \{x_j x_{j+1} \ldots x_n | 1 \le j \le n\} \cup \{\lambda\}, \tag{2}$$

and we define that $\mathcal{S}(\lambda) = \{\lambda\}$. The *dictionary* $\mathcal{D}(\boldsymbol{x})$ is defined as the set of all substrings of $\boldsymbol{x}$, that is,

$$\mathcal{D}(\boldsymbol{x}) = \{x_k x_{k+1} \ldots x_l \mid 1 \le k \le l \le n\} \cup \{\lambda\}. \tag{3}$$

A string $\boldsymbol{v} = v_1 v_2 \ldots v_m$ $(m \ge 1)$ with following three properties

$$\boldsymbol{v} \notin \mathcal{D}(\boldsymbol{x}), \tag{4}$$

$$v_1 \ldots v_{m-1} \in \mathcal{D}(\boldsymbol{x}), \tag{5}$$

$$v_2 \ldots v_m \in \mathcal{D}(\boldsymbol{x}) \tag{6}$$

is called a *Minimal Forbidden Word* (MFW) of $\boldsymbol{x}$. Note that $v_1 \ldots v_{m-1}$ and $v_2 \ldots v_m$ are λ in case of $m = 1$. The *antidictionary* of $\boldsymbol{x}$, denoted by $\mathcal{A}(\boldsymbol{x})$, is defined as the set of all MFWs of $\boldsymbol{x}$. For example, the antidictionary $\mathcal{A}(\boldsymbol{x})$ of $\boldsymbol{x} = 011021$ and $\mathcal{X} = \{0, 1, 2, 3\}$, is given by

$$\{3, 00, 12, 20, 22, 010, 101, 111, 210, 211\}. \tag{7}$$

Moreover, $\mathcal{A}(\boldsymbol{x})$ is classified into two classes, $\mathcal{A}_I(\boldsymbol{x})$ and $\mathcal{A}_L(\boldsymbol{x})$ [17]. For a string $\boldsymbol{x}$ of length n, $\mathcal{A}_I(\boldsymbol{x})$ is defined as

$$\mathcal{A}_I(\boldsymbol{x}) = \{\boldsymbol{v} | \boldsymbol{v} = \boldsymbol{u}a \in \mathcal{A}(\boldsymbol{x}), \boldsymbol{u} \in \mathcal{D}(\boldsymbol{x}^{n-1}), a \in \mathcal{X}\}, \tag{8}$$

and $\mathcal{A}_L(\boldsymbol{x}) = \mathcal{A}(\boldsymbol{x}) \backslash \mathcal{A}_I(\boldsymbol{x})$, respectively. For example, $\mathcal{A}_I(\boldsymbol{x})$ and $\mathcal{A}_L(\boldsymbol{x})$ of $\boldsymbol{x} = 011021$ are given by

$$\{3, 00, 12, 20, 22, 010, 101, 111\} \text{ and } \{210, 211\}, \tag{9}$$

respectively. We use a fanction $|\cdot|$ to represent not only the length of a string but also the size of a set.

The suffix tree $\mathbb{T}(\boldsymbol{x})$ is a tree structure that stores all elements of $\mathcal{S}(\boldsymbol{x})$ [23]. Let $\mathbb{T}_i$ be the suffix tree of $\boldsymbol{x}^i$. The string associated with the path from the root ρ to a node p in $\mathbb{T}(\boldsymbol{x})$ is called the *path-string* and is denoted by $\boldsymbol{w}(p)$. Notice that let $\boldsymbol{w}(\rho)$ be the null string λ. For any node p in $\mathbb{T}_i$, let $\mathcal{L}_i(p)$ be the set of all symbols that are associated with all edges sprouting from p, that is,

$$\mathcal{L}_i(p) = \{a \,|\, \boldsymbol{w}(p)a \in \mathcal{D}(\boldsymbol{x}^i), a \in \mathcal{X}\}. \tag{10}$$

A node p is an *internal* node if the size $|\mathcal{L}_i(p)| \ge 1$, otherwise p is an *external* node (or *leaf*), that is $|\mathcal{L}_i(p)| = 0$. An internal node p is called an *explicit* node if $|\mathcal{L}_i(p)| \ge 2$, otherwise p is called an *implicit* node, that is $|\mathcal{L}_i(p)| = 1$. Figure 1 shows suffix tree $\mathbb{T}(\boldsymbol{x})$ of $\boldsymbol{x} = 011021$. It consists of fourteen internal nodes and five leaves. White circles, black circles, and squares indicate explicit, implicit nodes and leaves, respectively. For example, for node p_2, $\boldsymbol{w}(p_2) = 1$ and $\mathcal{L}_6(p_2) = \{0, 1\}$. Note that for an MFW $\boldsymbol{u}a \in \mathcal{A}_I(\boldsymbol{x})$, there exists an internal node p such as $\boldsymbol{w}(p) = \boldsymbol{u}$, while for an MFW $\boldsymbol{v}a \in \mathcal{A}_L(\boldsymbol{x})$, there exists the leaf q with the shortest path length among all the leaves of $\mathbb{T}(\boldsymbol{x})$, where $\boldsymbol{w}(q) = \boldsymbol{v}$ [24].

For a node $p \ne \rho$ in $\mathbb{T}_i$, we can write $\boldsymbol{w}(p) = a\boldsymbol{v}$, where $a \in \mathcal{X}$ and $\boldsymbol{v} \in \mathcal{X}^*$. Let q be a node such that $\boldsymbol{w}(q) = \boldsymbol{v}$, and we establish a pointer from p to q, called a *suffix link*, and denoted by $\sigma(p)$. For a node r in $\mathbb{T}_i$, let d be an integer

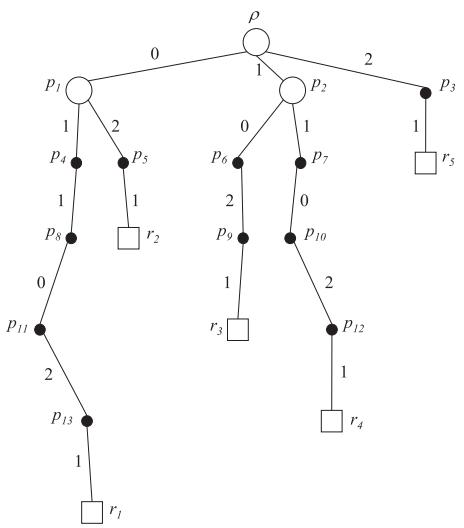


Fig. 1 Suffix tree $\mathbb{T}(\boldsymbol{x})$ of $\boldsymbol{x} = 011021$.

such as $0 \leq d \leq |\boldsymbol{w}(r)|$. For a fixed d, $\sigma_d(r)$ represents a node s such that $\boldsymbol{w}(s) \in \mathcal{S}(\boldsymbol{w}(r))$ and $|\boldsymbol{w}(s)| = d$ are satisfied. In other words, s is a node on suffix links from r to ρ such as $|\boldsymbol{w}(s)| = d$. For example, $\sigma(p_{10})$, $\sigma_1(p_{10})$, and $\sigma_3(p_{13})$ represent p_6, p_1, and p_9, respectively.

2.2 ECG Lossless Compression Based on the DCA

For a given binary string $\boldsymbol{x}$ of length n, DCA algorithm encodes $\boldsymbol{x}$ to $\boldsymbol{\gamma}$ by using $\mathcal{A}(\boldsymbol{x})$ with an off-line manner in linear time [12]. Suppose to have just read proper prefix $\boldsymbol{x}^i$ of $\boldsymbol{x}$. If a string $\boldsymbol{u}0$ (resp. $\boldsymbol{u}1$) $\in \mathcal{A}(\boldsymbol{x})$, such that $\boldsymbol{u}$ is a suffix of $\boldsymbol{x}^i$, then symbol x_{i+1} is not symbol 0 (resp. 1) since the alphabet is binary $\{0, 1\}$. Therefore, the following symbol x_{i+1} is surely symbol 1 (resp. 0). The DCA eliminates x_{i+1} to compress $\boldsymbol{x}$ since the next symbol x_{i+1} is turned out to be redundant or predictable. The DCA outputs a triplet $(\mathcal{A}(\boldsymbol{x}), \boldsymbol{\gamma}, n)$ as the codeword. The DCA uses a finite deterministic automaton, called *AD-automaton*, to find a proper MFW, that is $\boldsymbol{u}0$ (resp. $\boldsymbol{u}1$), efficiently [12]. The AD-automaton is constructed from $\mathcal{A}(\boldsymbol{x})$ or the subset. Figure 2 shows an AD-automaton of $\mathcal{A}(\boldsymbol{x}) = \{000, 111, 1010, 1101\}$. Circles and squares represent *states* and *sinks*, respectively. State q_0 represents the initial state. A state has two outgoing edges labeled symbol 0 and 1, respectively. For a state s and an input symbol $a \in \{0, 1\}$, a transition $(s, a) = t$ from state s to state t with a is implemented by traversing an outgoing edge labeled symbol a from s. For example, $(q_3, 0)$ and $(q_2, 1)$ is q_4 and q_3, respectively. In encoding process, transitions are implemented by using $\boldsymbol{x}$ starting from q_0. The output $\boldsymbol{\gamma}$ is given by using an AD-automaton as follows: if a state has two outgoing edges toward states, then the output symbol coincide with its input symbol; if a state has one outgoing edge toward a sink, then no symbol is output since a transition to a sink corresponds to an occurrence of an MFW [12]. To avoid trivial cases, we suppose that a state has at least one outgoing edge toward a state. For example, for $\boldsymbol{x} = 1100100$, $\boldsymbol{\gamma}$ is given by 1100 since the transitions is implemented as follows:

$$q_0 \to q_3 \to q_6 \to q_7 \to q_2 \to q_3 \to q_4 \to q_2. \qquad (11)$$

Symbols are output at states q_0, q_3, q_4.

To encode $\boldsymbol{x}$ of ECG data with an on-line manner, ECG data compression algorithms based on the DCA have been proposed [20], [22]. In [22], the EDCA algorithm and the EACDCA algorithm which is the EDCA applied to arithmetic coding have been proposed. Arithmetic coding is one of entropy codings (cf. [10]). Figure 3 shows diagrams of the EDCA and the EACDCA. The EDCA has the advantage of computational time, while the EACDCA has that of compression ratios. Both algorithms use $\mathcal{A}_l(\boldsymbol{t})$ of a proper prefix $\boldsymbol{x}^l$ $(= \boldsymbol{t})$ or a substring $\boldsymbol{t}$ of $\boldsymbol{x}$ of length l as training data instead of a whole input string $\boldsymbol{x}$ since each ECG wave is similar to the others. The antidictionary $\mathcal{A}_l(\boldsymbol{t})$ is constructed in the preprocessing. It was studied on the string length l needed to construct an antidictionary whose size is almost same as that of the whole string of ECG [20]. In Fig. 3, AC encoder and AC decoder represent an encoder and a decoder of adaptive arithmetic coding order-0 (cf. [10]), respectively. To improve compression ratios, the EACDCA encodes $\boldsymbol{\gamma}$ by using AC encoder. Let $N(p)$ and $N(p, a)$ be a number of transition times from state p and a number of transition times of edge labeled symbol a from p, where $N(p) = N(p, 0) + N(p, 1)$. Note that an initial value $N(p, a)$ is assigned by 1. The EACDCA encodes a probability $N(p, a)/N(p)$ by using AC encoder instead of outputting symbol a straightforwardly. In other words, the EACDCA uses an AD-automaton as a probabilistic model. A basis of the idea was first proposed by Ohkawa *et al.*, while their algorithm only works with an off-line manner [16].

2.3 ACDCA Algorithms

First, a set $\mathcal{V}_i(\boldsymbol{x}^i)$ is defined as a subset of $\mathcal{A}_l(\boldsymbol{x}^i)$, that is,

$$\mathcal{V}_i(\boldsymbol{x}^i) = \{\boldsymbol{v} | \boldsymbol{v} = \boldsymbol{u}a \in \mathcal{A}_l(\boldsymbol{x}^i), \boldsymbol{u} \in \mathcal{S}(\boldsymbol{x}^i), a \in \mathcal{X}\}. \qquad (12)$$

If $|\mathcal{V}_i(\boldsymbol{x}^i)| = |\mathcal{X}| - 1$ and $\boldsymbol{u}x_{i+1} \notin \mathcal{V}_i(\boldsymbol{x}^i)$ for any $\boldsymbol{u} \in \mathcal{S}(\boldsymbol{x}^i)$ are satisfied, then the symbol x_{i+1} is eliminated with DCA manner since x_{i+1} is predictable. However, straightforward implementations of dynamic DCA algorithms require worst case $O(n^2)$ time with respect to a string length n to calculate $|\mathcal{V}_i(\boldsymbol{x}^i)|$ since they need to update $\mathcal{A}_l(\boldsymbol{x}^i)$ whenever a new

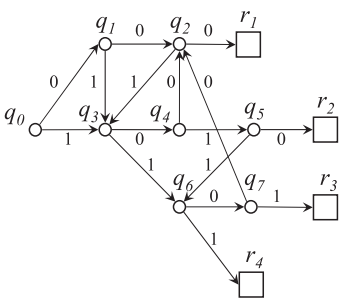


Fig. 2 AD-automaton of $\mathcal{A}_l(\boldsymbol{x}) = \{000, 111, 1010, 1101\}$.

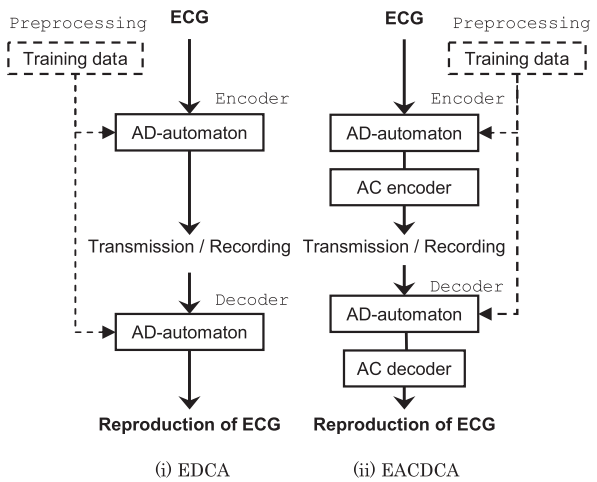


Fig. 3 Block diagrams of two ECG compression systems (EDCA and EACDCA).

Table 1 Relationship between α_i and the codeword of x_{i+1}.

case	output	conditions	notes
(a)	$(I, \mathsf{R}(x_{i+1}))$	$x_{i+1} \notin \mathcal{L}_i(\alpha_i)$	an MFW occurs.
(b)	none	$x_{i+1} \in \mathcal{L}_i(\alpha_i)$ **and** $\lvert\mathcal{L}_i(\alpha_i)\rvert = 1$	x_{i+1} is eliminated.
(c)	$\Pr(x_{i+1}\lvert\alpha_i)$	$x_{i+1} \in \mathcal{L}_i(\alpha_i)$ **and** $\lvert\mathcal{L}_i(\alpha_i)\rvert \geq 2$	α_i is an explicit node.

symbol is read [12], [14]–[16].

On the other hand, the ACDCA algorithms calculate $|\mathcal{V}_i(\boldsymbol{x}^i)|$ with an on-line manner in linear time [17]–[19]. The ACDCA algorithms use dynamic suffix trees to reduce computational time. A linear construction algorithm of dynamic suffix tree has been proposed by Ukkonen [25]. In the Ukkonen algorithm, a pointer to node in suffix tree, called *active point*, plays a key roll in linear time for the construction of the tree. An active point α_i of $\mathbb{T}_i$ is defined as follows.

Definition 1 (**active point).** *The active point α_i is the node p of $\mathbb{T}_i$ such that $\boldsymbol{w}(p)$ is the longest string in $(\mathcal{S}(\boldsymbol{x}^i) \cap \mathcal{D}(\boldsymbol{x}^{i-1}))$.*

In other words, $\boldsymbol{w}(\alpha_i)$ is the longest suffix of $\boldsymbol{x}^i$ that has a copy in $\mathcal{D}(\boldsymbol{x}^{i-1})$. For active point α_i and $i > 0$,

$$|\mathcal{V}_i(\boldsymbol{x}^i)| = |\mathcal{X}| - |\mathcal{L}_i(\alpha_i)| \tag{13}$$

holds [17]. From (13), we can calculate $|\mathcal{V}_i(\boldsymbol{x}^i)|$ by using $|\mathcal{X}|$ and $|\mathcal{L}_i(\alpha_i)|$.

Table 1 shows the relationship between α_i and the codeword of x_{i+1} in the ACDCA. In case-(a), I represents an interval of occurrence of case-(a) and $\mathsf{R}(x_{i+1})$ represents a rank of x_{i+1} in $\mathcal{X}$ $(0 \leq \mathsf{R}(x_{i+1}) \leq |\mathcal{X}| - 1)$. Let $\mathcal{R}_i$ be a set of the longest string $\boldsymbol{w}(p)c$ in $(\mathcal{S}(\boldsymbol{w}(\alpha_i)c) \cap \mathcal{D}(\boldsymbol{x}^{i-1}))$ or $\{c\}$ for each $c \in \mathcal{X}$. Strings of $\mathcal{R}_i$ are sorted in descending order of length either that of the following counter $\tilde{N}(c|p)$ in case-(c) if lengths are the same or in lexicographical order if those are the same. The rank $\mathsf{R}(x_{i+1})$ is the rank of the string which has x_{i+1} as the last symbol in $\mathcal{R}_i$. The rank $\mathsf{R}(x_{i+1})$ is used to convert x_{i+1} into a small integer to improve the compression ratio, and it can be simply determined in the Ukkonen algorithm. The details are described in [18]. The pair I and $\mathsf{R}(x_{i+1})$ are encoded by adaptive arithmetic coding using order-0 model. In case-(b), no symbol is output since x_{i+1} is eliminated from (13). In case-(c), we use the probability $\Pr(x_{i+1}|\alpha_i)$ to encode x_{i+1} by adaptive arithmetic coding order-0 model. For an explicit node p, let $\tilde{N}(c|p)$ be a counter that has the number of traversed times by an active point from p with symbol c. The probability $\Pr(x_{i+1}|\alpha_i)$ is given by $\tilde{N}(x_{i+1}|\alpha_i)/\sum_{c\in\mathcal{L}_i(\alpha_i)}\tilde{N}(c|\alpha_i)$.

3. Proposed Algorithm

The ACDCA algorithms work in an on-line manner with linear time and give better compression ratios than those of the DCA algorithms do. Moreover, those algorithms are able to deal with a string over a finite alphabet. However, the ACDCA algorithms require computational memory in proportional to the string length. On the other hand, the EDCA and the EACDCA work with constant computational memory, while they are able to deal with only a string on a binary alphabet.

In this section, we propose a new on-line ECG lossless compression algorithm based on the ACDCA and the EACDCA. The proposed algorithm works with constant computational memory in an on-line manner. To deal with ECG data over a finite alphabet, the proposed algorithm constructs its encoder and decoder from training data by using the ACDCA with an on-line manner.

3.1 Preliminaries

To reduce computational memory, we use a set of MFWs of restricted length less than or equal to $d + 1$. A set $\mathcal{W}_i(\boldsymbol{x}^i)$ is defined as a subset of elements of $\mathcal{V}_i(\boldsymbol{x}^i)$ whose length is less than or equal to $d + 1$, that is,

$$\mathcal{W}_i(\boldsymbol{x}^i) = \{\boldsymbol{w} | \boldsymbol{w} = za \in \mathcal{A}_I(\boldsymbol{x}^i), z \in \mathcal{S}(\boldsymbol{x}^i), |z| \leq d, a \in \mathcal{X}\}. \tag{14}$$

Next, we define *modified active point* μ_i to calculate $|\mathcal{W}_i(\boldsymbol{x}^i)|$ efficiently and show Proposition 1.

Definition 2 (**modified active point).** *For a fixed integer $d \geq 0$ and α_i,*

$$\mu_i = \begin{cases} \alpha_i & (|\boldsymbol{w}(\alpha_i)| \leq d), \\ \sigma_d(\alpha_i) & (|\boldsymbol{w}(\alpha_i)| > d). \end{cases} \tag{15}$$

Proposition 1. *For μ_i and $|\mathcal{W}_i(\boldsymbol{x}^i)|$, if $i > 0$, then*

$$|\mathcal{W}_i(\boldsymbol{x}^i)| = |\mathcal{X}| - |\mathcal{L}_i(\mu_i)|. \tag{16}$$

Proof. The proof is omitted here since it is described in the proof of Proposition 1 in [19], [28]. □

From (16), we can calculate $|\mathcal{W}_i(\boldsymbol{x}^i)|$ by using $|\mathcal{X}|$ and $|\mathcal{L}_i(\mu_i)|$. The proposed algorithm uses a subtree of $\mathbb{T}_i$, called *ST-automaton*, whose height is $d + 1$, to calculate $|\mathcal{L}_i(\mu_i)|$ efficiently. The subtree has the same root and consists of all internal nodes within height d and all edges sprouting from the internal nodes in $\mathbb{T}_i$. Figure 4 shows a ST-automaton of $\boldsymbol{x} = 0110211$ for $d = 2$. Break lines represent suffix links, and all the nodes except root ρ have a suffix link. A white circle has at least two edges and black circle has an edge. A modified active point μ_i traverses internal nodes of a ST-automaton. Note that μ_i does not traverse deepest edges which are at $d + 1$ in the ST-automaton since a length of $\boldsymbol{w}(\mu_i)$ does not exceed d from Definition 2. To avoid trivial cased, all the nodes have at least one edge.

Table 2 shows the relationship between μ_i and the codeword of x_{i+1} in the proposed algorithm. The proposed algorithm uses μ_i instead of α_i to encode x_{i+1} shown in Table 2, while the ACDCA uses α_i to encode x_{i+1} shown in Table 1. In case-(a), $\hat{I}$ represents an interval of occurrence of case-(a) and $\hat{\mathsf{R}}(x_{i+1})$ represents a rank of x_{i+1} in $\mathcal{X}$ $(0 \leq \hat{\mathsf{R}}(x_{i+1}) \leq |\mathcal{X}| - 1)$. In case-(b), no symbol is output since x_{i+1} is eliminated from (16). In case-(c), we use

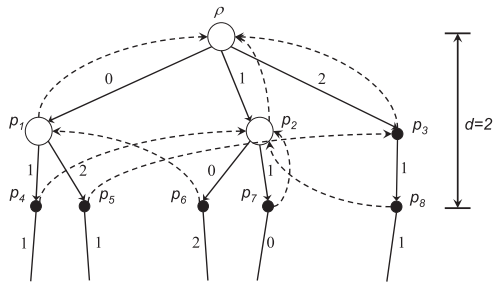


Fig. 4 ST-automaton for $\boldsymbol{x}$ = 0110211.

Table 2 Relationship between μ_i and the codeword of x_{i+1}.

case	output	conditions	notes
(a)	$(\hat{I}, \hat{\mathsf{R}}(x_{i+1}))$	$x_{i+1} \notin \mathcal{L}_i(\mu_i)$	an MFW occurs.
(b)	none	$x_{i+1} \in \mathcal{L}_i(\mu_i)$ **and** $\lvert\mathcal{L}_i(\mu_i)\rvert = 1$	x_{i+1} is eliminated.
(c)	$\Pr(x_{i+1}\lvert\mu_i)$	$x_{i+1} \in \mathcal{L}_i(\mu_i)$ **and** $\lvert\mathcal{L}_i(\mu_i)\rvert \geq 2$	μ_i is an explicit node.

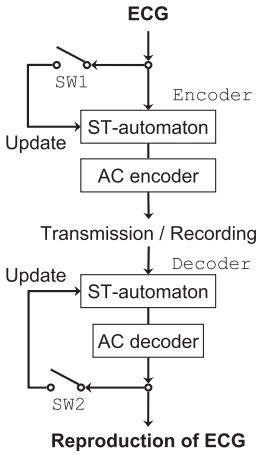


Fig. 5 Block diagram of the proposed compression system.

the probability $\Pr(x_{i+1}|\mu_i)$ to encode x_{i+1} by adaptive arithmetic coding order-0 model. The probability $\Pr(x_{i+1}|\mu_i)$ is given by $\hat{N}(x_{i+1}|\mu_i)/\sum_{c\in\mathcal{L}_i(\mu_i)}\hat{N}(c|\mu_i)$, where $\hat{N}(c|p)$ counts number of traversed times by a modified active point from p with symbol c.

3.2 Proposed ECG Compression System

Figure 5 shows a diagram of the proposed algorithm. Both ST-automatons are updated only if SW1 and SW2 are closed. To reduce computational memory of the ST-automatons to constant and to synchronize them, we use a fixed length l of prefix of ECG as training data $\boldsymbol{t}$ (= $\boldsymbol{x}^l$) to construct them.

A symbol x_{i+1} ($i < l$) is encoded by using a updating ST-automaton, that is a subtree of $\mathbb{T}_i$, while a symbol x_{j+1} ($j \geq l$) is encoded by using a fixed ST-automaton constructed from $\boldsymbol{x}^l$. To encode x_{j+1}, a pointer π_j to node of the fixed ST-automaton is used instead of μ_i in Table 2. The codeword of x_{j+1} is obtained by replacing μ_i with π_j in Table 2. Initial pointer π_l indicates μ_l, and π_{j+1} is determined from π_j and x_{j+1} as follows:

$$\pi_{j+1} = \begin{cases} (\pi_j, x_{j+1}), & (x_{j+1}\in\mathcal{L}_j(\pi_j), |\boldsymbol{w}(\pi_j)|<d) \\ (\sigma(\pi_j), x_{j+1}), & (x_{j+1}\in\mathcal{L}_j(\pi_j), |\boldsymbol{w}(\pi_j)|=d) \\ \eta. & (x_{j+1}\notin\mathcal{L}_j(\pi_j)) \end{cases} \quad (17)$$

Node η is a node (p, x_{j+1}) such that $x_{j+1} \in \mathcal{L}_j(p)$ and $\boldsymbol{w}(p)$ is the longest string among all nodes on the suffix links from $\sigma(\pi_j)$ to ρ. If there does not exist p such as $x_{j+1} \in \mathcal{L}_j(p)$, then η is ρ. By using suffix links of a ST-automaton, the node η is determined, simply. For example, for $\boldsymbol{z}$ = 0110023 and initial $\pi_0 = \rho$, transitions on a fixed ST-automaton shown in Fig. 4 are as follows:

$$\rho \xrightarrow{0} p_1 \xrightarrow{1} p_4 \xrightarrow{1} p_7 \xrightarrow{0} p_6 \xrightarrow{0} p_1 \xrightarrow{2} p_5 \xrightarrow{3} \rho. \quad (18)$$

In transitions $(p_6, 0)$ and $(p_5, 3)$, output of case-(a) in Table 2 occur, and in $(p_4, 1)$ and $(p_7, 0)$, case-(b) occur. In $(\rho, 0)$, $(p_1, 1)$, and $(p_1, 2)$, output of case-(c) occur.

Note that asymptotic optimality of the proposed algorithm for a stationary ergodic Markov source has been proved [29].

4. Experimental Results

In this section, we show the performance of the proposed algorithm by simulation results. In experiments, we used ECG files on the MIT-BIH Arrhythmia Database [21] since ECG is frequent measured and monitored for a patient having a heart disease. An ECG data which is sampled at a rate of 360 samples/s (one sample is represented as 16 bits (2 bytes) by padded 5 zeros of one sample with 11 bits resolution) and has $6.5 \cdot 10^5$ samples (1.3 Mbytes (about 30 min.)). To evaluate performance of the proposed algorithm compared with standard data compression applications such as bzip2 [26], the proposed algorithm uses one byte (= 8 bits) for a symbol since they also use one byte for a symbol. It is reported that the EDCA and the EACDCA used one bit for a symbol [22]. In other words, one sample of ECG data is represented by two symbols (= 2 bytes) in the proposed algorithm. A compression ratio is given by (compressed file-size)/(input file-size (=1.3 Mbytes)). In our experiments, the left-padded 5 zeros are not removed in preprocessing. The reason is that the proposed algorithm uses one byte for a symbol and entropy coding. By using entropy coding, length of codeword for a symbol depends on only probability calculated by number of traversed times with the symbol from a node of its ST-automaton even if the symbol includes the padded zeros.

Table 3 shows compression ratios of the proposed algorithm on ten files on the MIT-BIH Arrhythmia Database, along with the EDCA [22], the EACDCA [22], and bzip2 which is one of high-performance off-line lossless compression applications using the Burrows-Wheeler Transformation (BWT) [27] with entropy coding. In Table 3, we used l = 0.1 Mbyte as the training data length. The reason is

Table 3 Compression ratios ($l = 0.1$ Mbyte).

ECG file	proposed	EDCA	EACDCA	bzip2
100	**0.24**	0.31	0.26	0.26
101	**0.25**	0.35	0.27	0.27
102	**0.24**	0.32	0.26	0.26
103	**0.26**	0.34	0.29	0.27
104	**0.27**	0.37	0.29	0.28
105	0.31	0.37	0.34	**0.30**
200	**0.31**	0.38	0.33	**0.31**
201	**0.24**	0.32	0.26	**0.24**
202	**0.28**	0.33	0.30	**0.28**
203	0.35	0.43	0.39	**0.34**
Average	**0.27**	0.35	0.29	0.28

Table 4 Compression ratios ($l = 1.3$ Mbytes).

ECG file	proposed	bzip2	ACDCA
100	**0.23**	0.24	0.26
101	**0.24**	0.25	0.27
102	**0.23**	0.24	0.26
103	**0.24**	0.25	0.28
104	**0.26**	0.27	0.30
105	0.28	**0.27**	0.31
200	**0.29**	**0.29**	0.33
201	**0.22**	**0.22**	0.25
202	**0.25**	0.26	0.29
203	**0.32**	**0.32**	0.36
Average	**0.25**	0.26	0.29

that the EDCA and the EACDCA used $l = 0.1$ Mbyte in experiments [22]. To obtain the best results for the proposed algorithm with $l = 0.1$ Mbyte, we selected the optimal value of d, that is $d = 2$. It is reported that both the EDCA and the EACDCA with $l = 0.1$ Mbyte used unbounded d, that is $d = \infty$. In the experiments, bzip2 used 0.1 Mbyte as a block size B. The BWT needs at least $5 \cdot B$ bytes to store an array of symbols ($1 \cdot B$ bytes) and an array of their indices ($4 \cdot B$ bytes) in our experiments. Experimental results show that the proposed algorithm achieved better compression ratios for all files than the EDCA and the EACDCA. Moreover, the proposed algorithm achieved better compression ratios for 5 files and the same ratios for 3 files in 10 files than that of bzip2, and the average compression ratio of the proposed algorithm was better than that of bzip2. Note that bzip2 is not able to work in an on-line manner since it is an off-line compression algorithm, while the proposed algorithm works in an on-line manner. In file 105 and file 203, the compression ratios are worse than those of bzip2. The reason is that there are many noise signals in the latter of the files which do not appear in the training data.

Here, if we apply $d = 2$ to the EACDCA with $l = 0.1$ Mbyte, its average compression ratio rises up to 0.37 for the same files shown in Table 3. The EACDCA applied to $d = 2$ uses a set of MFWs whose length is at most 24 bits (= 3 bytes). Note that for a given d, the maximum length of MFWs is given by $d+1$ bytes from (14). The conditions $d = 2$ and $l = 0.1$ Mbyte are the same conditions of the proposed algorithm shown in Table 3. Experimental results show that compression ratios improve as alphabet size increases.

In our experiments on a 3.2 GHz Pentium 4 with 2 Gbytes memory, it took about 22.1 second (about 588 kbits/s) in average to finish encode an ECG file in the proposed algorithm. The proposed algorithm is able to work in real-time since an ECG sampling rate is about 6 kbits/s. With respect to computational memory of coders, ST-automatons used 260 kbytes in the average in the experiments, while bzip2 needs at least 500 kbytes for the BWT. Moreover, the average is less than that of the EACDCA since the average computational memory of AD-automatons is 900 kbytes [22] for the same database.

Table 4 shows that the best compression ratios of the proposed algorithm, bzip2, and the ACDCA (L-ACDCA [18]) in our experiments. To obtain the best results for the proposed algorithm, we selected the optimal values of d and l which are 3 and $l = 1.3$ Mbytes, respectively. The ACDCA uses $l = 1.3$ Mbytes and $d = \infty$. In the experiments, bzip2 used 0.9 Mbyte as a block size B to obtain best compression ratios. Experimental results show that the proposed algorithm achieved better compression ratios for 6 files and the same ratios for 3 files in 10 files than those of bzip2. Moreover, the proposed algorithm with $l = 1.3$ Mbytes and $d = 3$ achieved better compression ratios for all files than results of the ACDCA and the proposed algorithm with $l = 0.1$ Mbyte and $d = 2$ shown in Table 3. From the results of the ACDCA, it is needed to use a proper d to achieve a good compression ratio. On the other hand, the computational memory and time were worse than the results of the proposed algorithm with $l = 0.1$ Mbyte. The proposed algorithm required 2.5 Mbytes for a ST-automaton, while bzip2 needs at least 4.5 Mbytes for the BWT. It took 23.1 second (563 kbits/s) to finish encoding on average.

Figures 6, 7, and 8 show observed relation among computational memory of ST-automaton, training data length l, and impact of height d for ECG file 100, 101, and 102, respectively. Experimental results show that the memory is proportional to d and l. In the experiments, for a fixed l, the algorithm with $d = 4$ required the memory whose size is twice the memory with $d = 2$ on average. On the other hand, for a fixed d, the algorithm with $2l$ required the memory whose size is 1.68 times the memory with l. Therefore, to reduce the memory, decreasing d is more efficient than decreasing l. Figure 9, 10, and 11 show observed relation among compression ratio, training data length l, and impact of height d for ECG file 100, 101, and 102, respectively. Note that compression ratios with $d = 0$ and $d = 1$ were at least 15% worse than that with $d = 3$. In the experiments, the algorithm with $d = 2$ achieved the best compression ratios when l is less than 0.3 Mbyte on average, while that with $d = 3$ did them when l is more than 0.3 Mbyte on average. Moreover, for all d, compression ratios improved as l increased.

Therefore, a good compression ratio can be achieved by using $d = 3$ and increasing l, while the memory increases as l grows. There is a trade-off between compression ratio and computational memory for a given fixed d.

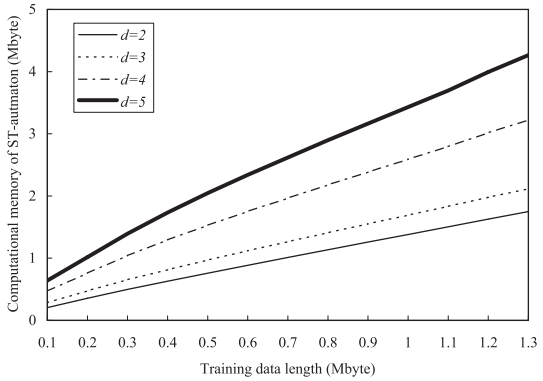


Fig. 6 Relationship between training data length l and computational memory of ST-automaton for fixed heights $d = 2, 3, 4, 5$ (ECG file 100).

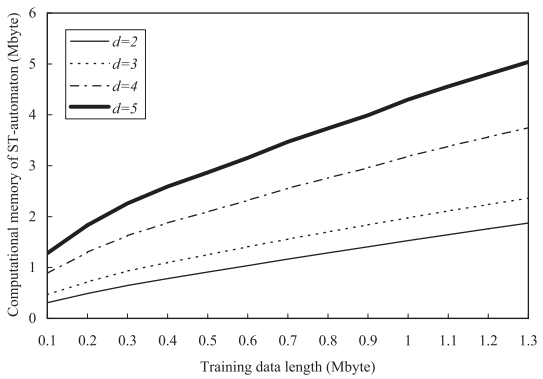


Fig. 7 Relationship between training data length l and computational memory of ST-automaton for fixed heights $d = 2, 3, 4, 5$ (ECG file 101).

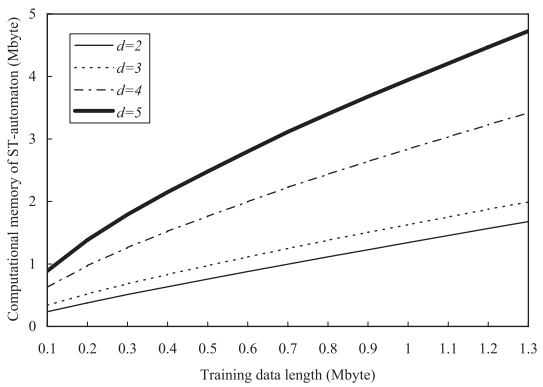


Fig. 8 Relationship between training data length l and computational memory of ST-automaton for fixed heights $d = 2, 3, 4, 5$ (ECG file 102).

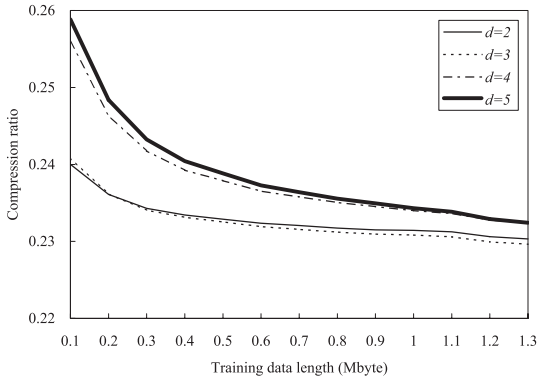


Fig. 9 Relationship between training data length l and compression ratio for fixed heights $d = 2, 3, 4, 5$ (ECG file 100).

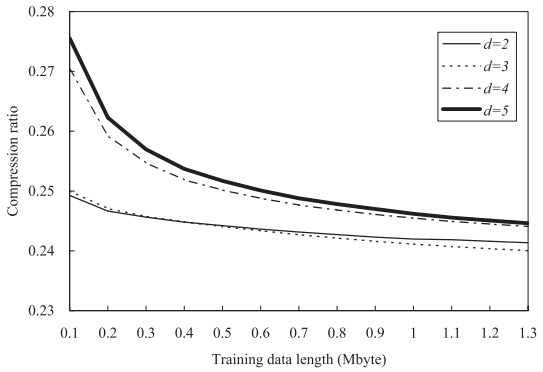


Fig. 10 Relationship between training data length l and compression ratio for fixed heights $d = 2, 3, 4, 5$ (ECG file 101).

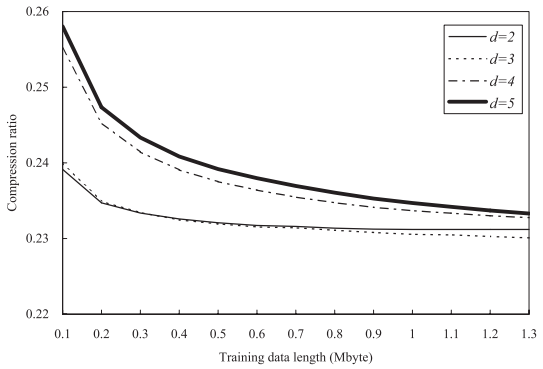


Fig. 11 Relationship between training data length l and compression ratio for fixed heights $d = 2, 3, 4, 5$ (ECG file 102).

5. Conclusion

In this paper, we proposed a new on-line ECG lossless compression using antidictionary codes. The proposed algorithm can handle a string over finite alphabet, while traditional algorithms using antidictionary codes deal with only a binary string. Moreover, the proposed algorithm can be implemented without a preprocessing and works with a constant computational memory.

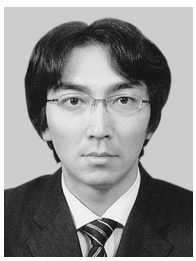
Experimental results showed that the proposed algorithm achieved better compression ratios for all files on the MIT-BIH Arrhythmia Database than those of the traditional algorithms. The proposed algorithm gives better compression ratios in average than those of bzip2 which is one of high-performance off-line lossless compression applications. Moreover, it was shown that the proposed algorithm needs 71% small computational memory in average coder size relative to the traditional algorithm. It was confirmed that the proposed algorithm works in real-time by simula-

tion results.

In future works, we plan to apply an antidictionary code using sliding windows [18] to ECG lossless compression for improving compression ratios.

References

[1] M. Kawasaki and R. Kohno, "A TOA based positioning technique of medical implanted devices," Third International Symposium on Medical Information and Communication Technologies (ISMICT2009), Montreal, Canada, Feb. 2009.
[2] F. Labeau, A.B. Tchana, and T. Le-Ngoc, "Enabling context aware clinical applications through ultra-wideband localization," Third International Symposium on Medical Information and Communication Technologies (ISMICT2009), Montreal, Canada, Feb. 2009.
[3] S. Nagamine and R. Kohno, "Design of communication model suitable for implanted body area networks," Third International Symposium on Medical Information and Communication Technologies (ISMICT2009), Montreal, Canada, Feb. 2009.
[4] S. Hanna, "Regulations and standards for wireless medical applications," Third International Symposium on Medical Information and Communication Technologies (ISMICT2009), Montreal, Canada, Feb. 2009.
[5] A.D. Jurik and A.C. Weaver, "Control, analysis, and visualization of body sensor streams," Third International Symposium on Medical Information and Communication Technologies (ISMICT2009), Montreal, Canada, Feb. 2009.
[6] S.M.S. Jalaleddine, C.G. Hutchens, R.D. Strattan, and W.A. Coberly, "ECG data compression technique—A unified approach," IEEE Trans. Biomed. Eng., vol.37, no.4, pp.329–343, April 1990.
[7] H. Morita and K. Kobayashi, "Data compression of ECG based on the edit distance algorithms," IEICE Trans. Inf. & Syst., vol.E76-D, no.12, pp.1443–1453, Dec. 1993.
[8] M. Ishijima, S. Shin, G.H. Hostetter, and J. Skalansky, "Scan-along polygonal approximation for data compression of electrocardiograms," IEEE Trans. Biomed. Eng., vol.BME-30, no.11, pp.723–729, Nov. 1983.
[9] M. Nelson and J.-L. Gailly, The Data Compression Book, M & T Books, 1996.
[10] A. Moffat and A. Turpin, Compression and coding algorithm, Kluwer Academic Publishers, 2002.
[11] M. Crochemore, F. Mignosi, A. Restivo, and S. Salemi, "Text compression using antidictionaries," Automata, Languages and Programming, 26th International Colloquium (ICALP'99), pp.261–270, July 1999.
[12] M. Crochemore, F. Mignosi, A. Restivo, and S. Salemi, "Data compression using antidictionaries," Proc. IEEE, vol.88, no.11, pp.1756–1768, Nov. 2000.
[13] J. Ziv and A. Lempel, "A universal algorithm for sequential data compression," IEEE Trans. Inf. Theory, vol.IT-23, no.3, pp.337–343, May 1977.
[14] M. Crochemore and G. Navarro, "Improved antidictionary based compression," XII International Conference of the Chilean Computer Science Society (SCCC'02), pp.7–13, Nov. 2002.
[15] M. Crochemore, C. Epifanio, R. Grossi, and F. Mignosi, "A trie-based approach for compacting automata," Proc. Combinatorial Pattern Matching 15th Annual Symp. (CPM'04), vol.3109, pp.145–158, July 2004.
[16] N. Ohkawa, K. Harada, and H. Yamamoto, "Data compression by arithmetic coding and source modeling based on the anti-dictionary method," IEICE Technical Report, IT2005-49, July 2005.
[17] T. Ota and H. Morita, "On the on-line arithmetic coding based on antidictionaries with linear complexity," Proc. 2007 IEEE Int'l Symp. on Inform. Theory (ISIT2007), pp.86–90, June 2007.
[18] T. Ota and H. Morita, "On the sliding window variations of antidictionary data compression using dynamic suffix trees," Proc. 2008 Int'l Symp. on Information Theory and its Applications (ISITA2008), pp.1105–1110, Dec. 2008.
[19] T. Ota and H. Morita, "Adaptive antidictionary data compression using a set of restricted length of minimal forbidden words," IEICE Technical Report, IT2008-99, March 2009.
[20] T. Ota and H. Morita, "One-pass ECG lossless compression using antidictionaries," IEICE Trans. Fundamentals (Japanese Edition), vol.J87-A, no.9, pp.1187–1195, Sept. 2004.
[21] MIT-BIH Arrhythmia Database, http://www.physionet.org/physiobank/database/mitdb/
[22] T. Ota and H. Morita, "On-line electrocardiogram lossless compression using antidictionary-based methods," Third International Symposium on Medical Information and Communication Technologies (ISMICT2009), Montreal, Canada, Feb. 2009.
[23] D. Gusfield, Algorithms on Strings, Trees, and Sequences, Cambridge University Press, 1997.
[24] T. Ota and H. Morita, "On the construction of an antidictionary with linear complexity using the suffix tree," IEICE Trans. Fundamentals, vol.E90-A, no.11, pp.2533–2539, Nov. 2007.
[25] E. Ukkonen, "On-line construction of suffix-trees," Algorithmica, vol.14, pp.249–260, 1995.
[26] bzip2, http://www.bzip.org/
[27] M. Burrows and D.J. Wheeler, "A block-sorting lossless data compression algorithm," SRC Research Report, May 1994.
[28] T. Ota and H. Morita, "On the adaptive antidictionary code using minimal forbidden words with constant lengths," Proc. 2010 Int'l Symp. on Information Theory and its Applications (ISITA2010), pp.72–77, Oct. 2010.
[29] T. Ota and H. Morita, "Asymptotic optimality of antidictionary codes," Proc. 2010 IEEE Int'l Symp. on Inform. Theory (ISIT2010), pp.101–105, June 2010.



Takahiro Ota received the B.E. and Ph.D. degrees from the University of Electro-Communications, Tokyo, Japan, in 1993 and 2007, respectively. In 1997, he joined Nagano Prefectural Institute of Technology, Nagano, Japan, first a Lecturer at Department of Electronic Engineering, where from 2009, he is an Associate Professor. His current research interests are in information theory, and source coding.

Hiroyoshi Morita received the B.E., M.E., and D.E. degrees from Osaka University, Osaka, Japan, in 1978, 1980 and 1983, respectively. In 1983, he joined Toyohashi University of Technology, Aichi, Japan as a Research Associate in the School of Production System Engineering. In 1990, he joined the University of Electro-Communications, Tokyo, Japan, first an Assistant Professor at the Department of Computer Science and Information Mathematics, where from 1992, he was an Associate Professor. Since 1995 he has been with the Graduate School of Information Systems, where from 2005, he is a Professor. He was Visiting Fellow at the Institute of Experimental Mathematics, University of Essen, Essen, Germany during 1993–1994. His research interests are in combinatorial theory, information theory, and coding theory, with applications to the digital communication systems.