

# On-line Electrocardiogram Lossless Compression Using Antidictionary-Based Methods

Takahiro OTA<sup>a</sup>, Hiroyoshi Morita<sup>b</sup>

<sup>a</sup>Department of Electronic Engineering, Nagano Prefectural Institute of Technology, Nagano, Japan

<sup>b</sup>Graduate School of Information Systems, University of Electro-Communications, Tokyo, Japan

## Abstract

This paper proposes on-line electrocardiogram (ECG) lossless data compression based on antidictionaries. An antidictionary is the set of all words of minimal length that never appear in the string. The proposed methods use the coders constructed by a training sequence of constant length of ECG, so that they work with an on-line manner in constant computational space. Their effectiveness is demonstrated by simulation results.

**Keywords:** Electrocardiogram, Lossless Data Compression, Antidictionary

## Introduction

Data compression is a scheme that reduces the data size by deleting the redundancy of data, and lossless data compression allows the original data to be reconstructed exactly from the compressed data. Data Compression is particularly useful in systems where resources are scarce, e.g., limited bandwidth in communication systems and the capacity of storage systems. A wide variety of data compression algorithms have been proposed for inherently digital data such as text and digitized audio, image, video and biomedical data (e.g., [1]-[4]). 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.

In 1999, Crochemore *et al.* proposed an off-line lossless data compression algorithm using an *antidictionary* of an input binary string [5, 6]. An antidictionary for a given string is the set of all words of minimal length, called minimal forbidden words, which never appear in the string. Their method, called Data Compression using Antidictionary (DCA), was applied to the Calgary Corpus and it was shown to perform as well as popular compression algorithms such as the Lempel-Ziv algorithms [7]. It was also proved that the DCA algorithm achieves a compression rate for a balanced binary source that is equal to its entropy rate. In 2002, an on-line data compression and detecting arrhythmia of *ElectroCardioGram* (ECG) was proposed [8]. Experimental results showed that this coding algorithm achieved better compression ratios than those of DCA and LZ algorithms [9]. Original DCA algorithm can handle only binary alphabet, while in 2004 Crochemore *et al.* proposed extensions of the antidictionary coding to any strings

over any finite alphabet [10]. Moreover, Ohkawa *et al.* [11] and the authors [12] showed an automaton and a tree model used as an encoder of the DCA algorithms provide efficient probabilistic models for arithmetic coding (e.g., [1, 3]) by simulation results.

The DCA algorithm and its extensions require an antidictionary to construct their encoders and decoders. Two construction algorithms of the antidictionary using Directed Acyclic Word Graph (DAWG) and suffix trie were proposed [13, 6]. The DAWG-based construction algorithm works in  $O(n)$  time, while the trie-based algorithm works in  $O(n^2)$  time with respect to the input string length  $n$ . In 2005, the construction algorithm using a suffix tree that is a compacted suffix trie was proposed [14] and it is proved that it works in  $O(n)$  time [15, 16]. In 2007, Fukae *et al.* proposed an on-line construction algorithm of an antidictionary using suffix array [17].

Despite their compression performance, the DCA algorithms are far less commonly applied than compression algorithms like the LZ algorithms. One of the main reasons is that the DCA algorithms usually work in an off-line manner, that is, the static scheme. The dynamic, that is an on-line manner, DCA algorithms have the advantage of compression ratios and reading a give string only once. The properties are needed to handle streaming files such as video on network and apply DCA algorithms to real-time systems. However, the straightforward implementations of on-line DCA algorithms require worst case  $O(n^2)$  time because they need to update an antidictionary and their coders whenever a new symbol is read. In 2006, an on-line linear compression algorithm based on the DCA algorithm was proposed [18, 19]. The proposed algorithm, called Arithmetic Coding Based on DCA (ACDCA), gives 20% improvement in average compressed file size relative to the DCA algorithm and achieves compression ratios as well as those of `gzip2` in which BW transformation [20] is used [21]. Moreover, the ACDCA constructs an efficient probabilistic model as the encoder. Nishiara *et al.* applied the model to branch prediction algorithm in computer architecture. It was shown that the algorithm achieves almost same prediction accuracy as a traditional predictor and the execution time are shortened from 1/50 to 1/5400 by computer simulation [22]. Moreover, an analysis of the size of antidictionary on memory-less binary sources and average length of minimal forbidden words on a stationary ergodic source were shown [23], [24].

In this paper, we present brief introduction to ACDCA and propose two on-line ECG lossless compression algorithms based on antidictionaries.

## Basic Definitions and Notations

Let  $X = \{0, 1, 2, \dots, m-1\}$  be finite source alphabet and  $X^*$  be the set of all finite strings over  $X$ , including the null string of length zero, denoted by  $\lambda$ . For a string  $\mathbf{x} = x_1x_2\dots x_n$  in  $X^*$ , let  $\mathbf{x}^i$  be a *prefix* of  $\mathbf{x}$  of length  $i$ . Let  $\mathcal{S}(\mathbf{x})$  be the set of all *suffixes* of  $\mathbf{x}$  including  $\lambda$ , that is,

$$\mathcal{S}(\mathbf{x}) = \{x_i \dots x_n \mid 1 \leq i \leq n\} \cup \{\lambda\}. \quad (2)$$

The *dictionary*  $\mathcal{D}(\mathbf{x})$  is defined as the set of all substrings of  $\mathbf{x}$  of length  $n$  including  $\lambda$ , that is,

$$\mathcal{D}(\mathbf{x}) = \{x_i \dots x_j \mid 1 \leq i \leq j \leq n\} \cup \{\lambda\} \quad (3)$$

A string  $\mathbf{v} = v_1v_2\dots v_k$  with the following properties

$$\begin{aligned} \mathbf{v} &\notin \mathcal{D}(\mathbf{x}) \\ v_1v_2\dots v_{k-1} &\in \mathcal{D}(\mathbf{x}) \\ v_2v_3\dots v_k &\in \mathcal{D}(\mathbf{x}) \end{aligned} \quad (4)$$

is called a *minimal forbidden word* (MFW) of  $\mathbf{x}$ . The *antidictionary* of  $\mathbf{x}$ , denoted by  $\mathcal{A}(\mathbf{x})$ , is defined as the set of all MFWs of  $\mathbf{x}$ . As an example,  $\mathcal{A}(\mathbf{x})$  for  $\mathbf{x} = 0110120$  is given by

$$\{00, 02, 21, 22, 010, 111, 112, 201, 1011\}. \quad (5)$$

Moreover,  $\mathcal{A}(\mathbf{x})$  is classified into two classes,  $\mathcal{A}_I(\mathbf{x})$  and  $\mathcal{A}_L(\mathbf{x})$  [19].

$$\mathcal{A}_I(\mathbf{x}) = \{\mathbf{v} \mid \mathbf{v} = \mathbf{ua} \in \mathcal{A}(\mathbf{x}), \mathbf{u} \in \mathcal{D}(\mathbf{x}^{n-1}), a \in X\}$$

$$\mathcal{A}_L(\mathbf{x}) = \{\mathbf{v} \mid \mathbf{v} = \mathbf{ua} \in \mathcal{A}(\mathbf{x}), \mathbf{u} \notin \mathcal{D}(\mathbf{x}^{n-1}), a \in X\}$$

In (5),  $\mathcal{A}_I(\mathbf{x}) = \{00, 02, 21, 22, 010, 111, 112, 1011\}$  and  $\mathcal{A}_L(\mathbf{x}) = \{201\}$ .

## Review of the DCA Algorithms

To encode an input string  $\mathbf{x}$ , the DCA algorithm eliminates symbols of  $\mathbf{x}$  by using  $\mathcal{A}(\mathbf{x})$ . Suppose to have just read proper  $\mathbf{x}^i$  of a binary string  $\mathbf{x}$ . If string  $\mathbf{u}0$  belongs to  $\mathcal{A}(\mathbf{x})$  (resp.  $\mathbf{u}1$ ), such as  $\mathbf{u}$  belongs to  $\mathcal{S}(\mathbf{x})$ , then symbol  $x_{i+1}$  is not symbol 0 (resp. 1). The symbol  $x_{i+1}$  is surely symbol 1 (resp. 0). Therefore, the DCA algorithm eliminates  $x_{i+1}$  because in advance  $x_{i+1}$  can be turned out to be redundant or predictable. In case of alphabet is finite instead of binary, the DCA algorithm works as well. Let  $\mathcal{V}_i(\mathbf{x})$  be a subset of  $\mathcal{A}(\mathbf{x})$ , that is,

$$\mathcal{V}_i(\mathbf{x}) = \{\mathbf{v} \mid \mathbf{v} = \mathbf{ua} \in \mathcal{A}_I(\mathbf{x}), \mathbf{u} \in \mathcal{S}(\mathbf{x}^i), a \in X\}. \quad (6)$$

For an arbitrary  $i$ , if  $|\mathcal{V}_i(\mathbf{x})| = m - 1$ , then  $x_{i+1}$  is eliminated in the DCA algorithm, where  $|\mathcal{V}_i(\mathbf{x})|$  is the size of  $\mathcal{V}_i(\mathbf{x})$ . In (6), we use  $\mathcal{A}_I(\mathbf{x})$  instead of  $\mathcal{A}(\mathbf{x})$  because elements of  $\mathcal{A}_L(\mathbf{x})$  are no useful for eliminating a symbol since the string  $\mathbf{u}$  such that  $\mathbf{ua}$  belongs to  $\mathcal{A}(\mathbf{x})$  is a suffix of  $\mathbf{x}$  (e.g., [19]). For example, Table 1 shows the relationship between output string of DCA algorithm and  $|\mathcal{V}_i(\mathbf{x})|$ , where  $\mathbf{x} = 0110120$ ,  $\mathcal{A}_I(\mathbf{x}) = \{00, 02, 21, 22,$

$010, 111, 112, 1011\}$  and  $X = \{0, 1, 2\}$ . As shown in Table 1, a symbol can be eliminated if  $|\mathcal{V}_i(\mathbf{x})| = 2$  since  $m = 3$  from (6). The off-line DCA algorithm output the 3-tuples in (7) as its codeword.

$$(\mathcal{A}_I(\mathbf{x}), \boldsymbol{\gamma}, |\mathbf{x}|) \quad (7)$$

Note that  $|\mathbf{x}|$  represents the length of string  $|\mathbf{x}|$ . In Table 1,  $\boldsymbol{\gamma}$  and  $|\mathbf{x}|$  are given by 01 and 7, respectively. The DCA algorithm uses a finite deterministic automaton, called *AD-automaton*, to find proper MFWs, that is  $|\mathcal{V}_i(\mathbf{x})|$ , efficiently. The AD-automaton is constructed from  $\mathcal{A}_I(\mathbf{x})$  or the subset of  $\mathcal{A}_I(\mathbf{x})$ .

Table 1 An example of encoding of the DCA algorithm, where  $\mathbf{x} = 0110120$ ,  $\mathcal{A}_I(\mathbf{x}) = \{00, 02, 21, 22, 010, 111, 112, 1011\}$  and  $m = 3$ .

Input string	Output string	$ \mathcal{V}_i(\mathbf{x}) $	MFWs
$\mathbf{x}^0 = \lambda$	$\lambda$		
$\mathbf{x}^1 = 0$	0	0	
$\mathbf{x}^2 = 01$	0	2	00, 02
$\mathbf{x}^3 = 011$	01	1	010
$\mathbf{x}^4 = 0110$	01	2	111, 112
$\mathbf{x}^5 = 01101$	01	2	00, 02
$\mathbf{x}^6 = 011012$	01	2	010, 1011
$\mathbf{x}^7 = 0110120 (= \mathbf{x})$	01 (= $\boldsymbol{\gamma}$ )	2	21, 22

## ACDCA

The traditional DCA algorithms [5, 6, 10-12] need quadratic computational time  $O(n^2)$  to encode and decode  $\mathbf{x}$  in on-line manner. On the other hands, the ACDCA algorithm<sup>18, 19)</sup> works with linear time  $O(n)$  time in an on-line manner. To reduce computational time, the ACDCA algorithm uses dynamic suffix trees. A *suffix tree* [25] is useful for searching for a pattern in sequences such as DNA sequences and texts since it stores all suffixes of a given string. A linear on-line construction algorithm of suffix tree  $\mathbf{T}(\mathbf{x})$  of a given string  $\mathbf{x}$  is proposed by Ukkonen [26]. In the Ukkonen algorithm, a pointer to node in suffix tree, called *active point*, plays a key roll in  $O(n)$  time for the construction of the tree. An active point  $\alpha_i$  is defined as the following. The *path-string*  $\mathbf{w}(p)$  represents a labeled string on the suffix tree from the root node to a node  $p$ .

*Definition 1 (active point):* An active point  $\alpha_i$  in  $\mathbf{T}(\mathbf{x}^i)$  is the node having path-string  $\mathbf{s}$  such that the longest string in  $(\mathcal{S}(\mathbf{x}^i) \cap \mathcal{D}(\mathbf{x}^{i-1}))$ .

For active point  $\alpha_i$ , the following Theorem 1 holds.

*Theorem 1:* (Equation (15) in [19]) For active point  $\alpha_i$  and  $\mathcal{V}_i(\mathbf{x}^i)$ ,

$$|\mathcal{V}_i(\mathbf{x}^i)| = m - |\mathbf{L}(\alpha_i)|$$

holds, where  $\mathbf{L}(\alpha_i)$  represents the set of the first symbols of labeled string on all edges sprouting from  $\alpha_i$ . In other words,  $|\mathbf{L}(\alpha_i)|$  represents the number of the edges sprouting from  $\alpha_i$ . From Theorem 1, if  $|\mathbf{L}(\alpha_i)| = 1$  is satisfied, then  $x_{i+1}$  can be eliminated since  $|\mathcal{V}_i(\mathbf{x}^i)| = m - 1$ . To improve compression ra-

tios, the ACDCA algorithm applies adaptive arithmetic code to remained symbols. Table 2 shows codewords of  $x_{i+1}$  of the ACDCA algorithm.

In case-(a),  $I$  represents an interval of MFW occurrence. The pair  $I$  and  $x_{i+1}$  are encoded by adaptive arithmetic coding using order-0 model. In case-(b), no codeword is output since  $x_{i+1}$  is eliminated. In case-(c),  $Pr(x_{i+1}|\alpha_i)$  is used to encode  $x_{i+1}$  by adaptive arithmetic code order-0 model. The probability  $Pr(x_{i+1}|\alpha_i)$  is given by  $N(x_{i+1}|\alpha_i) / \sum_{c \in L(\alpha_i)} N(c|\alpha_i)$ , where  $N(b|p)$  is a counter which has the number of traversed times by an active point from node  $p$  with symbol  $b$ . The methods of improvement of compression ratios of case-(a) and case-(c) are proposed [21, 27].

Table 3 shows the compression results of the proposed on-line linear algorithm [21] (ACDCA) on the Calgary corpus, which is one of well-known bench mark archives for data compression, along with an on-line DCA[6], non-adaptive high-performance compression scheme labeled BW [20], and a popular compression application `gzip` [31].

Table 2 - Codewords of  $x_{i+1}$  of the ACDCA algorithm.

Case	Codewords	Relationship between $\alpha_i$ and $x_{i+1}$
(a)	$(I, x_{i+1})$	$x_{i+1} \notin L(\alpha_i)$
(b)	none	$x_{i+1} \in L(\alpha_i)$ and $ L(\alpha_i) =1$
(c)	$Pr(x_{i+1} \alpha_i)$	$x_{i+1} \in L(\alpha_i)$ and $ L(\alpha_i)  \geq 2$

Table 3 - Compression ratios

file	ACDCA	DCA	BW	gzip
bib	0.26	0.32	0.26	0.26
book1	0.34	0.38	0.31	0.41
book2	0.28	0.35	0.27	0.34
geo	0.63	0.78	0.56	0.67
news	0.33	0.43	0.32	0.38
obj1	0.51	0.61	0.50	0.48
obj2	0.32	0.45	0.33	0.33
paper1	0.32	0.40	0.32	0.35
paper2	0.33	0.39	0.31	0.36
pic	0.12	0.14	0.10	0.11
progc	0.32	0.40	0.32	0.34
progl	0.22	0.28	0.23	0.23
progp	0.22	0.28	0.22	0.23
trans	0.19	0.24	0.20	0.20
Average	0.31	0.39	0.30	0.34

Note that compression ratio is given by compressed file size / original file size. Experimental results show ACDCA achieved better compression ratios for 3 files and the same for 4 files compared with BW. The average result shows that ACDCA achieved almost same result of that of BW.

## An On-line ECG Lossless Compression

ECG is one of biomedical data, and numerous lossy data compression algorithms of ECG have been proposed [28]. However, from the point of view of biomedical data, an on-

line lossless compression is needed not to lose any significant features of the ECG signals. Figure 1 shows a schematic representation of normal ECG wave. Figure 2 shows an example of seven ECG waves. In Figure 2, the third wave from left is an arrhythmia and others are Normal Sinus Rhythm (NSR). As shown in Figure 2, each ECG wave is an almost periodic waveform and 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. On the other hands, ACDCA can be applied to ECG, however, it is difficult to handle an extremely long data such as ECG since ACDCA requires computational space proportional to the data size.

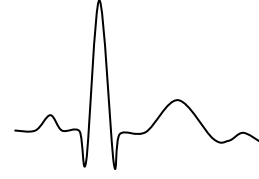


Figure 1 - A schematic representation of NSR.

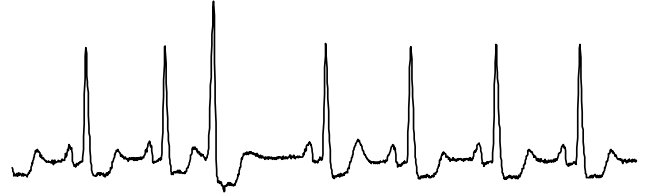


Figure 2 - An example of seven ECG waves.

In this section, we propose two on-line lossless ECG compression algorithms based on antidictionaries with constant computational space. To restrict computational space to constant, we construct an antidictionary and its encoder (AD-automaton) from ECG data of proper constant length called training data. We studied on the length of training data [9, 29] needed to construct the antidictionary whose size is the almost same as that of entire data of ECG by using results of coupon collector's problems (e.g., [32]). An ECG consists of many NSRs and a few arrhythmias, so that a sequence of NSRs on ECG is used as training data in our experiments.

Figure 3 shows the diagram of the proposed systems. Since a size of antidictionary is proportional to alphabet size  $|X|$  [30], both of training data and ECG used in proposed systems are represented by binary alphabet. In Figure 3, AC encoder and AC decoder represent an encoder and a decoder of adaptive arithmetic coding order-0 using the method proposed by Ohkawa *et al.* [11], respectively. Let  $w$ ,  $x$  be training data and ECG data. An AD-automaton is constructed from  $w$  in pre-process. In encoding and decoding process, an MFW can be occurred since  $A_I(w)$  is the almost same as  $A_I(x)$  but is not the same as  $A_I(x)$ . If an MFW occurs, then a transition of AD-automaton is restarted at the initial state. For ECG data, both of EDCA and EACDCA output the 2-tuples as shown in (8), (9), respectively. Let  $AC(s)$  be the codeword of  $s$  of adaptive arithmetic coding order-0 using the method [11].

$$(\gamma, I) \tag{8}$$

$$(AC(\gamma), I) \quad (9)$$

In (8) and (9),  $\gamma$  represents output of  $x$  in the DCA algorithm using  $A_I(\mathbf{w})$ , and  $I$  represents a sequence of interval lengths which are occurrences of MFWs. Thus if  $I=I_1I_2\dots I_k$ , then  $|x|=I_1+I_2+\dots+I_k$  holds. Note that only the last value  $I_k$  may be the length from the index of  $x$  at the occurrence of  $I_{k-1}$  to the index of the last symbol in  $x$ . In our experiments,  $I_j$  is represented by a binary representation of positive integer proposed by Elias.<sup>3)</sup> EDCA has the advantage of computational time, while EACDCA has that of compression ratios.

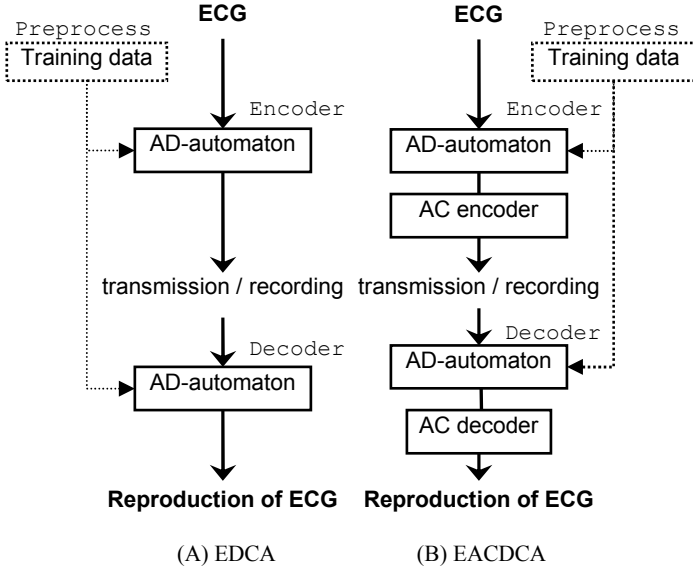


Figure 3 - Block diagram of ECG compression systems

Next, we show the performance of the proposed algorithm by simulation results. In experiment, we used an ECG data which is sampled at a rate of 360 samples/s (one sample is represented as 16-bit by padded 5 zero of one sample with 11-bit resolution) and has 650,000 samples (about 30 min.). The training data has 50,000 samples (about 138 sec.). Table 4 shows the compression results of the EDCA and EACDCA on the files on MIT-BIH Arrhythmia Database [34], along with the DCA, OM04 [9] and a popular compression application `gzip` [31]. For each ECG data, DCA uses the antidiictionary of entire ECG data, while OM04 uses an antidiictionary of training data. Both of antidiictionaries consist of MFWs whose maximum length is less than 32-bit (2 samples). Note that an MFW used in both EDCA and EACDCA has no limit on the length. DCA works in an off-line manner while OM04, `gzip` and proposed algorithms work in on-line manner.

Experimental results show EDCA and EACDCA achieved better compression ratios about 10% and 16% than `gzip`, respectively, and 1% and 7% than OM04, respectively. In our experiments on a 3.2 GHz Pentium 4 with 2 Gbytes memory, it took about 0.7 second (15 Mbits/s) in average to finish encode an ECG file in EDCA and about 1.8 second (6 Mbits/s) in EACDCA. The results show that both of proposed algorithms can implement to compress ECG files for real-time in an on-

line manner since sampling bit-rate of ECG is about 6 kbits/s. In our experiments, EDCA and EACDCA consumed 0.5 Mbytes and 0.9 Mbytes for AD-automaton in average.

## Conclusion

In this paper, we presented an efficient on-line ECG lossless data compression algorithms based on antidiictionaries. Experiments confirm that the proposed algorithms give better compression ratios than traditional algorithms. Moreover, the proposed algorithms are fast and memory-efficient due to a limited size of training data.

Table 4 - Compression ratios of ECG files

ECG file	EDCA (proposed)	EACDCA (proposed)	DCA	OM04	gzip
100	0.31	0.25	0.39	0.33	0.39
101	0.35	0.27	0.37	0.36	0.42
102	0.32	0.26	0.34	0.33	0.41
103	0.34	0.29	0.38	0.35	0.43
104	0.36	0.29	0.39	0.38	0.45
105	0.38	0.34	0.38	0.38	0.47
106	0.38	0.32	0.40	0.38	0.48
107	0.44	0.40	0.42	0.44	0.59
Ave.	0.36	0.30	0.38	0.37	0.46

## Acknowledgment

This research was partially supported by the Ministry of Education, Science, Sports and Culture Japan, Grant-in-Aid for Scientific Research (C), 19560370, 2007-8.

## References

- [1] Moffat A, Turpin A, *Compression and Coding Algorithm*, Kluwer Academic Publishers. 2002.
- [2] Nelson M, Gailly J-L, *The Data Compression Book*, M & T Books. 1996.
- [3] Society of Information Theory and its Applications, *Source Coding -Lossless Data Compression-*, Baifukan. 1998. (in Japanese)
- [4] Society of Information Theory and its Applications, *Source Coding -Lossy Data Compression-*, Baifukan. 2000. (in Japanese)
- [5] Crochemore M, Mignosi F, Restive A, S. Salemi, Text compression using antidiictionaries, *26th International Colloquium on Automata, Languages and Programming (ICALP1999)(LNCS 1644, Springer)*. 1999; 261-270.
- [6] ---, Data compression using antidiictionaries, *Proc. of the IEEE*. 2000; 88, 11, 1756-1768.
- [7] Ziv J, Lempel A, A universal algorithm for sequential data compression, *IEEE Trans. on Inform. Theory*. 1977; 23, 3, 337-343.

- [8] Ota T, Morita H, DCA lossless compression and detecting arrhythmia of ECG, *Proc. of the 25th Symp. on Inform. Theory and its Applications (SITA2002)*. 2002; II, 551-554. (in Japanese)
- [9] ---, One-pass ECG lossless compression using antidictionaries, *IEICE Trans. on Fundamentals (Japanese Edition)* 2004; J87-A, 9, 1187-1195.
- [10] Crochemore M, Epifanio C, Grossi R, Mignosi F, A trie based approach for compacting automata, *Proc. of the Combinatorial Pattern Matching 15th Annual Symp.* 2004; 3109, 145-158.
- [11] Ohkawa N, Harada K, Yamamoto H, Data compression by arithmetic coding and source modeling based on the anti-dictionary method, *IEICE Technical Report, IT2005-49*. 2005. (in Japanese)
- [12] Ota T, Morita H, Data compression using source model on antidictionary tree, *IEICE Technical Report, IT2005-87* 2006. (in Japanese)
- [13] Crochemore M, Mignosi F, Restivo A, Automata and forbidden words, *Information Processing Letters*. 1998; 67, 3, 111-117.
- [14] Ota T, Morita H, Construction antidictionary of a binary string using suffix tree, *IEICE Technical Report, IT2005-43*. 2005. (in Japanese)
- [15] ---, On the construction of an antidictionary of a binary string with linear complexity, *Proc. of 2006 IEEE Int'l. Symp. on Inform. Theory (ISIT2006)*. 2006; 2343-2347.
- [16] ---, On the construction of an antidictionary with linear complexity using the suffix tree, *IEICE Trans. on Fundamentals*. 2007; E90-A, 11, 2533-2539.
- [17] Fukae H, Morita H, Dynamic construction of antidictionary using suffix array, *Proc. of the 30th Symp. on Inform. Theory and its Applications*. 2007; 383-387. (in Japanese)
- [18] Ota T, Morita H, On-line arithmetic coding based on antidictionaries, *Proc. of the 29th Symp. on Inform. Theory and its Applications (SITA2006)*. 2006; 2, 513-516. (in Japanese)
- [19] ---, On the on-line arithmetic coding based on antidictionaries with linear complexity, *Proc. of 2007 IEEE Int'l. Symp. on Inform. Theory (ISIT2007)*. 2007; 86-90.
- [20] Burrows M, Wheeler D.J, A block-sorting lossless data compression algorithm, *SRC Research Report*. 1994.
- [21] Ota T, Morita H, On the sliding window variations of antidictionary data compression using dynamic suffix trees, *Proc. of 2008 Int'l Symp. on Inform Theory and its Applications (SITA2008)*. 2008; 1105-1110.
- [22] Nishiara M, Morita H, Ota T, Branch prediction based on antidictionary tree, *IEICE Technical Report, CAS2007-38*. 2007. (in Japanese)
- [23] Fayolle J, Analysis of the size of antidictionary in DCA, *Proc. of the Combinatorial Pattern Matching (CPM2008)*. 2008; 5029, 107-117.
- [24] Ota T, Morita H, On the length of minimal forbidden words on a stationary ergodic source, *Proc. of the 31st Symp. on Inform. Theory and its Applications 2008*; 688-691. (in Japanese)
- [25] Gusfield D, *Algorithms on Strings, Trees, and Sequences*, Cambridge Univ. Press. 1997.
- [26] Ukkonen E, On-line construction of suffix-trees, *Algorithmica*. 1995; 14, 3, 249-260.
- [27] Ota T, Morita H, Modeling based on antidictionaries for adaptive data compression, *Proc. of the 30th Symp. on Inform. Theory and its Applications (SITA2007)* 2007; 388-391. (in Japanese)
- [28] Jalaaliddine S.M.S, Hutchens C.G, Strattan R.D, Coberly W.A, ECG data compression technique –a unified approach-, *IEEE Trans. on Biomed. Eng* 1990; 37, 4, 329-343.
- [29] Ota T, Antidictionary data compression using dynamic suffix trees, *Ph.D thesis, The University of Electro-Communications*. 2007.
- [30] Mignosi F, Restivo A, Sciortino M, Words and forbidden factors, *Theoretical Computer Science*. 2002; 273, 99-117.
- [31] Gailly J-L, *gzip*. 2002; <http://www.gzip.org/>
- [32] Graham L, Kunuth D.E, Patashnik O, *Concrete Mathematics: A Foundation for Computer Science*, Addison-Wesley Pub. 1998.
- [33] Elias P, Universal codeword sets and representations of the integers, *IEEE Trans. Inform. Theory*. 1975; IT-21, 2, 194-203.
- [34] MIT-BIH Arrhythmia Database, <http://www.physionet.org/physiobank/database/mitdb/>

#### Address for correspondence

**a)** Takahiro Ota, Department of Electronic Engineering, Nagano Prefectural Institute of Technology, 813-8 Shimonogo, Ueda, Nagano, 386-1211 Japan.  
E-mail: ota@pit-nagano.ac.jp

**b)** Hiroyoshi Morita, Graduate School of Information Systems, University of Electro-Communications, 1-5-1 Chofugaoka, Chofu, Tokyo, 182-8585 Japan.  
E-mail: morita@is.uec.ac.jp