Hindawi Publishing Corporation
Journal of Sensors
Volume 2012, Article ID 539638, 20 pages
doi:10.1155/2012/539638

Research Article

An Adaptive Lossless Data Compression Scheme for Wireless Sensor Networks

Jonathan Gana Kolo,[1] S. Anandan Shanmugam,[1] David Wee Gin Lim,[1] Li-Minn Ang,[2] and Kah Phooi Seng[3]

[1] *Department of Electrical and Electronics Engineering, The University of Nottingham, Malaysia Campus, Jalan Broga, Selangor Darul Ehsan, 43500 Semenyih, Malaysia*
[2] *School of Engineering, Edith Cowan University, Joondalup, WA 6027, Australia*
[3] *School of Computer Technology, Sunway University, 5 Jalan Universiti, Bandar Sunway, Selangor, 46150 Petaling Jaya, Malaysia*

Correspondence should be addressed to Jonathan Gana Kolo, keyx1jgk@nottingham.edu.my

Received 4 July 2012; Revised 10 September 2012; Accepted 10 September 2012

Academic Editor: Eugenio Martinelli

Energy is an important consideration in the design and deployment of wireless sensor networks (WSNs) since sensor nodes are typically powered by batteries with limited capacity. Since the communication unit on a wireless sensor node is the major power consumer, data compression is one of possible techniques that can help reduce the amount of data exchanged between wireless sensor nodes resulting in power saving. However, wireless sensor networks possess significant limitations in communication, processing, storage, bandwidth, and power. Thus, any data compression scheme proposed for WSNs must be lightweight. In this paper, we present an adaptive lossless data compression (ALDC) algorithm for wireless sensor networks. Our proposed ALDC scheme performs compression losslessly using multiple code options. Adaptive compression schemes allow compression to dynamically adjust to a changing source. The data sequence to be compressed is partitioned into blocks, and the optimal compression scheme is applied for each block. Using various real-world sensor datasets we demonstrate the merits of our proposed compression algorithm in comparison with other recently proposed lossless compression algorithms for WSNs.

1. Introduction

Wireless sensor networks (WSNs) are suitable for large scale data gathering and they have become so increasingly important for enabling continuous monitoring in many fields. WSNs have find application in areas such as environmental monitoring, industrial monitoring, health and wellness monitoring, seismic and structural monitoring, inventory location monitoring, surveillance, power monitoring, factory and process automation, object tracking, precision agriculture, disaster management, and equipment diagnostics [1–5].

Sensor nodes in WSNs are generally self-organized and they communicate with each other wirelessly to perform a common task. The nodes are deployed in large number and scattered randomly in an ad-hoc manner in the sensor field. Each node is equipped with battery, wireless transceiver, microprocessors, sensors, and memory. Once deployed, the sensor nodes form a network through short-range wireless communication. Data collected by each sensor node is transferred wirelessly to the sink either directly or through multihop communication.

Technological advances in microelectromechanical systems (MEMS) in the recent past have lead to the production of very small size sensor nodes. The tiny size has placed serious resource limitations on the nodes ranging from a finite power supply, limited bandwidth for communication, limited processing speed, to limited memory and storage space. Besides the size, other stringent sensor node constraints include but are not limited to the following: extremely low power consumption; ability to operate in high density; must be cheap (low production cost) and be

Table 1: Huffman coding Table A.

b_i	h_i	d_i
0	00	0
1	01	$-1, +1$
2	11	$-3, -2, +2, +3$
3	101	$-7, \ldots, -4, +4, \ldots, +7$
4	1001	$-15, \ldots, -8, +8, \ldots, +15$
5	10001	$-31, \ldots, -16, +16, \ldots, +31$
6	100001	$-63, \ldots, -32, +32, \ldots, +63$
7	1000001	$-127, \ldots, -64, +64, \ldots, +127$
8	10000001	$-255, \ldots, -128, +128, \ldots, +255$
9	1000000000	$-511, \ldots, -256, +256, \ldots, +511$
10	10000000010	$-1023, \ldots, -512, +512, \ldots, +1023$
11	10000000011	$-2047, \ldots, -1024, +1024, \ldots, +2047$
12	10000000100	$-4095, \ldots, -2048, +2048, \ldots, +4095$
13	10000000101	$-8191, \ldots, -4096, +4096, \ldots, +8191$
14	10000000110	$-16383, \ldots, -8192, +8192, \ldots, +16383$

Table 2: Huffman coding Table B.

b_i	h_i	d_i
0	1101111	0
1	11010	$-1, +1$
2	1100	$-3, -2, +2, +3$
3	011	$-7, \ldots, -4, +4, \ldots, +7$
4	111	$-15, \ldots, -8, +8, \ldots, +15$
5	10	$-31, \ldots, -16, +16, \ldots, +31$
6	00	$-63, \ldots, -32, +32, \ldots, +63$
7	010	$-127, \ldots, -64, +64, \ldots, +127$
8	110110	$-255, \ldots, -128, +128, \ldots, +255$
9	110111011	$-511, \ldots, -256, +256, \ldots, +511$
10	110111001	$-1023, \ldots, -512, +512, \ldots, +1023$
11	1101110101	$-2047, \ldots, -1024, +1024, \ldots, +2047$
12	1101110100	$-4095, \ldots, -2048, +2048, \ldots, +4095$
13	1101110000	$-8191, \ldots, -4096, +4096, \ldots, +8191$
14	11011100011	$-16383, \ldots, -8192, +8192, \ldots, +16383$

Table 3: Huffman coding Table C.

b_i	h_i	d_i
0	1001	0
1	101	$-1, +1$
2	00	$-3, -2, +2, +3$
3	01	$-7, \ldots, -4, +4, \ldots, +7$
4	11	$-15, \ldots, -8, +8, \ldots, +15$
5	10001	$-31, \ldots, -16, +16, \ldots, +31$
6	100001	$-63, \ldots, -32, +32, \ldots, +63$
7	1000001	$-127, \ldots, -64, +64, \ldots, +127$
8	10000001	$-255, \ldots, -128, +128, \ldots, +255$
9	1000000000	$-511, \ldots, -256, +256, \ldots, +511$
10	10000000010	$-1023, \ldots, -512, +512, \ldots, +1023$
11	10000000011	$-2047, \ldots, -1024, +1024, \ldots, +2047$
12	10000000100	$-4095, \ldots, -2048, +2048, \ldots, +4095$
13	10000000101	$-8191, \ldots, -4096, +4096, \ldots, +8191$
14	10000000110	$-16383, \ldots, -8192, +8192, \ldots, +16383$

Table 4: Decision regions for our proposed ALDC scheme.

F region	Code option
$F \leq 3n$	2-Huffman Table ALEC
$3n < F \leq 12n$	3-Huffman Table ALEC
$12n < F$	2-Huffman Table ALEC

dispensable; be autonomous and operate unattended; and be adaptive to the environment [6].

Due to the hardware constraints mentioned above, wireless sensor nodes can only be equipped with a limited power source. In addition, the replacement of batteries for sensor nodes is virtually impossible for most applications since the nodes are often deployed in large numbers into harsh and inaccessible environments. Thus, the lifetime of WSN shows a strong dependence on battery lifetime. It is therefore important to carefully manage the energy consumption of each sensor node subunit in order to maximize the network lifetime of WSN. Furthermore, wireless sensor nodes are also constrained in terms of processing and memory. Therefore, software designed for use in WSNs should be lightweight and the computational requirements of the algorithms should be low for efficient operation in WSNs.

Sensor nodes in WSN consume energy during sensing, processing, and transmission. But typically, the energy spent by a sensing node in the communication module for data transmission and reception is more than the energy for processing [1–4, 7–13]. One significant approach to conserve energy and maximize network lifetime in WSN is through the use of efficient data compression schemes [5, 8]. Data compression schemes can be used to reduce the amount of information being exchanged in a network resulting in a saving of power. This savings due to compression directly translate into lifetime extension for the network nodes [14]. Both the local single node that compresses the data as well as the intermediate routing nodes benefits from handling less data [15].

In order to use WSNs most effectively, efficient compression schemes should be employed that not only reduce the size of the streaming data but also require minimal resources to perform the compression. Our aim in this paper is to accomplish this for continuous data collection applications in WSNs by exploiting temporal correlation using a local data compression scheme which has been shown to significantly improve WSN energy savings in real-world deployments [15]. A careful study of local data compression algorithms proposed in the literature for WSNs shows that most of the algorithms cannot dynamically adjust to changes in the source data statistics. Consequently, the compression performance obtained by the algorithms is not optimal. We therefore propose in this paper an adaptive lossless data compression (ALDC) scheme for WSN. The algorithm has the ability to adapt to changes in the source data statistics to maximize performance. The proposed ALDC algorithm compresses block of sampled data at a time using multiple code options adaptively. Our proposed ALDC algorithm operates in one pass and can be applied to multiple data types.

Table 5: Main characteristic of the datasets.

Deployment name	Node ID	Symbolic name	Number of samples	Time interval	
				From day	To day
LUCE	84	LU84	64913	23-Nov-06	17-Dec-06
HES-SO FishNet	101	FN101	12652	09-Aug-07	31-Aug-07
Le Génépi	20	LG20	21523	04-Sep-07	03-Oct-07

Table 6: Entropy of the original and residual data sets.

Data set	H	H_d
LU84 TEMP	10.07	4.05
FN101 TEMP	10.26	5.10
LG20 TEMP	10.25	6.82
LU84 RH	9.92	5.70
FN101 RH	9.61	5.73
LG20 RH	10.76	7.61
Seismic data	6.79	3.91

The rest of this paper is organized as follows. Section 2 discusses related work. Section 3 presents our proposed ALDC algorithms. In Section 4, the proposed algorithm is evaluated and compared with recent lossless compression algorithms using real-world WSN data. Finally, we conclude the paper in Section 5.

2. Related Work

Energy is typically more limited in WSNs than in other wireless networks because of the nature of the sensing devices and the difficulties in recharging or changing their batteries. The ability of data compression to provide energy efficiency rests on the favourable trade off between computational energy and transmission energy as recognized in the literature. Any data compression scheme designed for use in WSNs should be lightweight, and the computational requirements of the algorithms should be low for efficient operation due to WSNs constraints in terms of hardware, energy, processing, and memory. For these reasons, researchers have therefore designed and developed various compression algorithms specifically for WSNs. There are two general approaches for data compression in WSNs. One is the distributed data compression approach and the other is the local data compression approach. The distributed data compression approach exploits the high spatial correlation in data from fixed sensors node in dense networks. Some of the main techniques under this approach include distributed source coding (DSC) [16, 17], distributed transform coding (DTC) [18, 19], distributed source modeling (DSM) [20, 21], and compressed sensing (CS) [22]. The distributed compression approach however conserves energy at the expense of information loss in the source data and for this reason will not be considered. Instead the local data compression approach that takes advantage of the temporal correlation that exist in sampled sensor data to perform its compression locally on each sensor node will be considered. Some of the proposed local data compression algorithms based on temporal correlation in WSNs include: lossy algorithms (lightweight temporal compression (LTC) [14], K-RLE [23], differential pulse code modulation-based optimization (DPCM-Optimization) [24]; lossless algorithms (sensor node LZW (S-LZW) [15], Lossless Entropy Compression (LEC) [3], modified adaptive Huffman compression scheme [4], median-predictor-based data compression (MPDC) [25], two-modal transmission (TMT) [26]). The precision required by some application domains demands sensor nodes with very high accuracy that cannot tolerate measured data being corrupted by the compression process. Thus, in this section, we will focus on lossless local data compression algorithms.

The authors in [15] introduced a lossless compression algorithm called S-LZW which is an adapted version of LZW [27] designed specifically for resource constrained sensor nodes. It uses adaptive dictionary techniques with dynamic code length. The dictionary structure allows the algorithm to adapt to changes in the input and to take advantage of repetition in the sensed data. However, the algorithm suffers from the growing dictionary problem and its compression efficiency still needs to be improved.

In [3], the authors introduced Huffman coding into wireless sensor nodes. Their simple lossless entropy compression (LEC) algorithm which was based on static Huffman coding exploits the temporal correlation that exist in sensor data to compute a compressed version using a small dictionary, the size of the ADC resolution. The algorithm was particularly suitable for computational and memory resource constrained sensor nodes. The algorithm is static. Hence, the algorithm cannot adapt to changes in the source data statistics. In the paper [4], the proposed algorithm was a modified version of the classical adaptive Huffman coding. The algorithm does not require prior knowledge of the statistics of the source data and compression is performed adaptively based on the temporal correlation that exists in the source data. The drawback of this algorithm is that it is computationally intensive. In [25], the authors propose a compression algorithm that uses median predictor to decorrelate the sensed data. The proposed algorithm is simple and can be implemented in a few lines of code and uses the LEC compression table. The algorithm has similar compression complexity as LEC but lower compression efficiency. Since the LEC algorithm outperforms it, the algorithm will not be used for comparison with our algorithm.

In [26], the authors proposed a scheme called two-modal transmission (TMT) for predictive coding. In the first modal transmission, called compressed mode, the compressed bits of error terms falling inside the interval $[-R, R]$

Table 7: Compression performance comparison between ALDC with other recent lossless compression schemes.

Data set	Compression ratio (CR) %			
	LEC	S-LZW	ALDC with block size of 32	ALDC with block size of 48
LU84 TEMP	70.81	48.99	73.87	73.94
FN101 TEMP	65.39	30.35	67.44	67.48
LG20 TEMP	53.83	22.02	56.86	56.90
LU84 RH	62.86	31.24	65.50	65.54
FN101 RH	62.95	36.27	66.28	66.33
LG20 RH	48.67	21.93	52.80	52.87
Seismic data	69.72	43.33	73.41	73.38

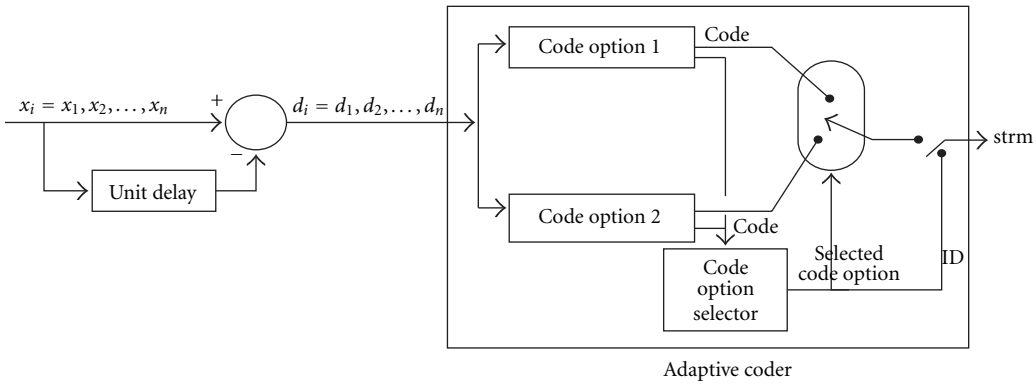

Figure 1: Functional block diagram of ALDC algorithm using the brute-force approach.

are transmitted. In the second modal transmission, called noncompressed mode, the original raw data of error terms falling outside the interval $[-R, R]$ are transmitted instead without compression. The modified predictive coding based on the two-modal transmission approach solved the problem of decreased coding efficiency due to the low performance of large error terms prediction. A second-order linear predictor was employed. The sink node was responsible for computing the coefficient values of the linear predictor. Arithmetic coding was chosen as the coding scheme. The authors applied the optimal M-based alphabet. The drawback of this compression algorithm is that it is computationally intensive. As such, to implement the scheme in WSNs, the sink node, which is not energy-limited, searches for the optimal predictor's coefficients, the optimal bound R and the optimal M for M-based alphabet coding. These optimal parameters are then transmitted to other sensor nodes to enable them to perform predictive coding based on the two-modal transmission algorithm.

The LEC algorithm is simple, and it requires low amount of memory for its execution. It has low computational complexity and gives the best lossless compression ratio performance till date. But, the LEC algorithm cannot adapt to changing correlation in sensor-measured data. Hence, the compression ratio obtained and by extension the energy saving obtainable is not optimal. This therefore gives room for improvement. We in this paper, therefore propose a new lossless data compression algorithm for WSNs called Adaptive Lossless Data Compression (ALDC) algorithm. Our proposed algorithm adapts to changes in the source data statistics to maximize compression performance. Our proposed ALDC algorithm operates in one pass using multiple code options adaptively and can be applied to multiple data types. With this improvement, our proposed ALDC algorithm outperforms the LEC algorithm.

3. Adaptive Lossless Data Compression Algorithm

In this section, we describe our proposed adaptive lossless data compression (ALDC) algorithm. Adaptive compression schemes allow compression to dynamically adjust to a changing source. Our proposed ALDC Scheme performs compression losslessly using two adaptive lossless entropy compression (ALEC) code options adaptively. The two ALEC code options, namely 2-Huffman Table ALEC and 3-Huffman Table ALEC, were originally presented in our article titled "An Efficient Lossless Adaptive Compression Algorithm for Wireless Sensor Networks." The 2-Huffman Table ALEC and the 3-Huffman Table ALEC are both adaptive coding scheme that adaptively uses two Huffman tables and three Huffman tables, respectively. The Huffman tables used by the two ALEC code options are given in Tables 1, 2, and 3. The Huffman tables were designed and arrived at after working with many real-world wireless sensor node datasets with varied levels of correlation. While 2-Huffman Table ALEC uses Huffman Coding Table A and Huffman Coding Table B, 3-Huffman Table ALEC uses Huffman Coding Table A, Huffman Coding Table B and Huffman

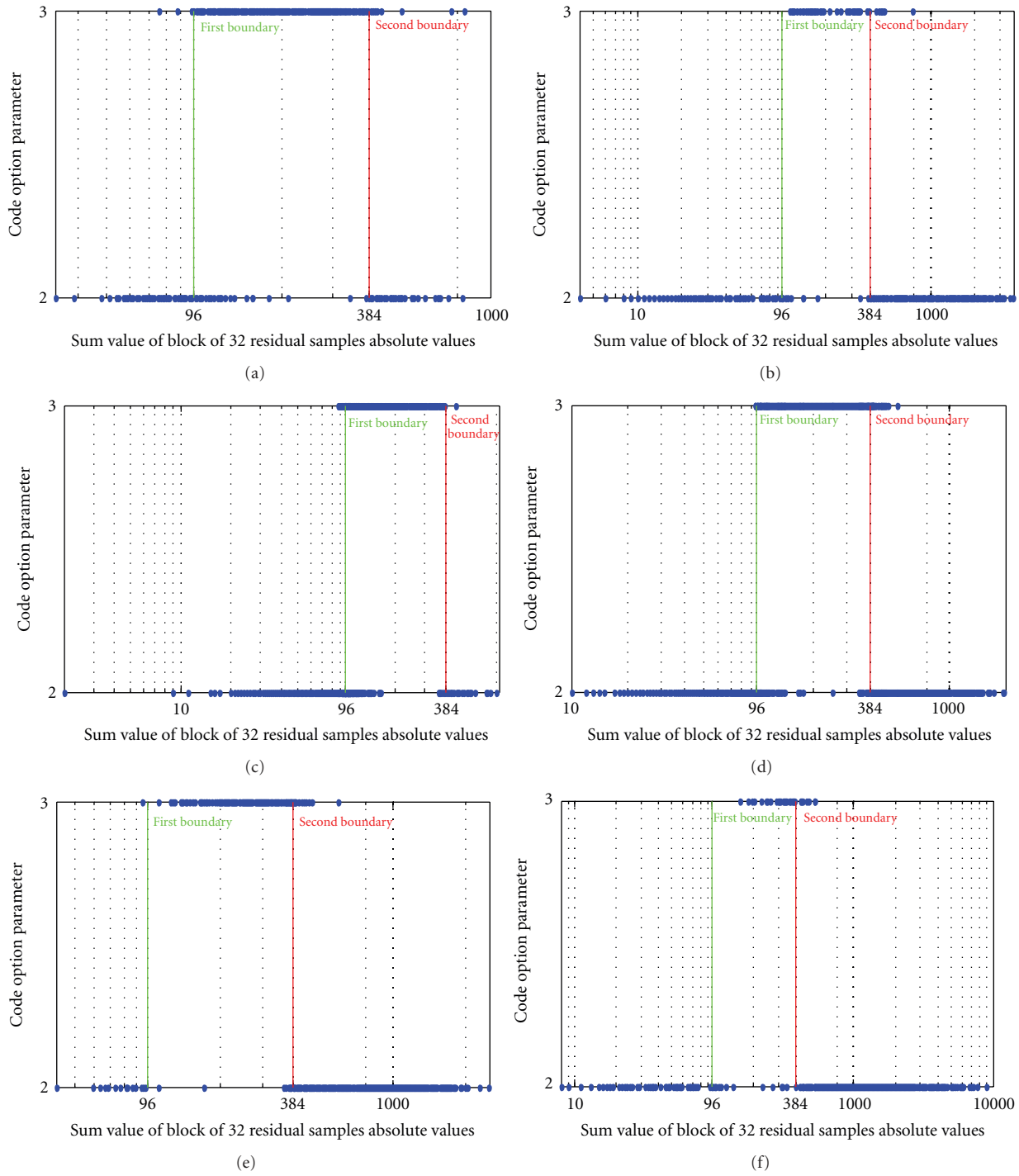

Figure 2: Code Option Parameter versus Sum value of Block of 32 residual samples absolute values for (a) FN101 temperature data set, (b) FN101 RH data set, (c) LU84 temperature data set, (d) LU84 RH data set, (e) LG20 temperature data set, and (f) LG20 RH data set.

Coding Table C. The two ALEC code options compresses block of sampled data at a time. While the 2-Huffman Table ALEC encodes block of sampled data in accordance with the pseudocode in Algorithm 1, the 3-Huffman Table ALEC encodes block of sampled data in accordance with the pseudocode in Algorithm 2. The pseudocode of the encode function called by both Algorithm 1 and Algorithm 2 is given in Algorithm 3. The encode function encodes each d_i as a bit sequence c_i composed of two parts h_i and l_i (i.e., $c_i = h_i * l_i$):

$$l_i = (\text{Index})|_{b_i}, \tag{1}$$

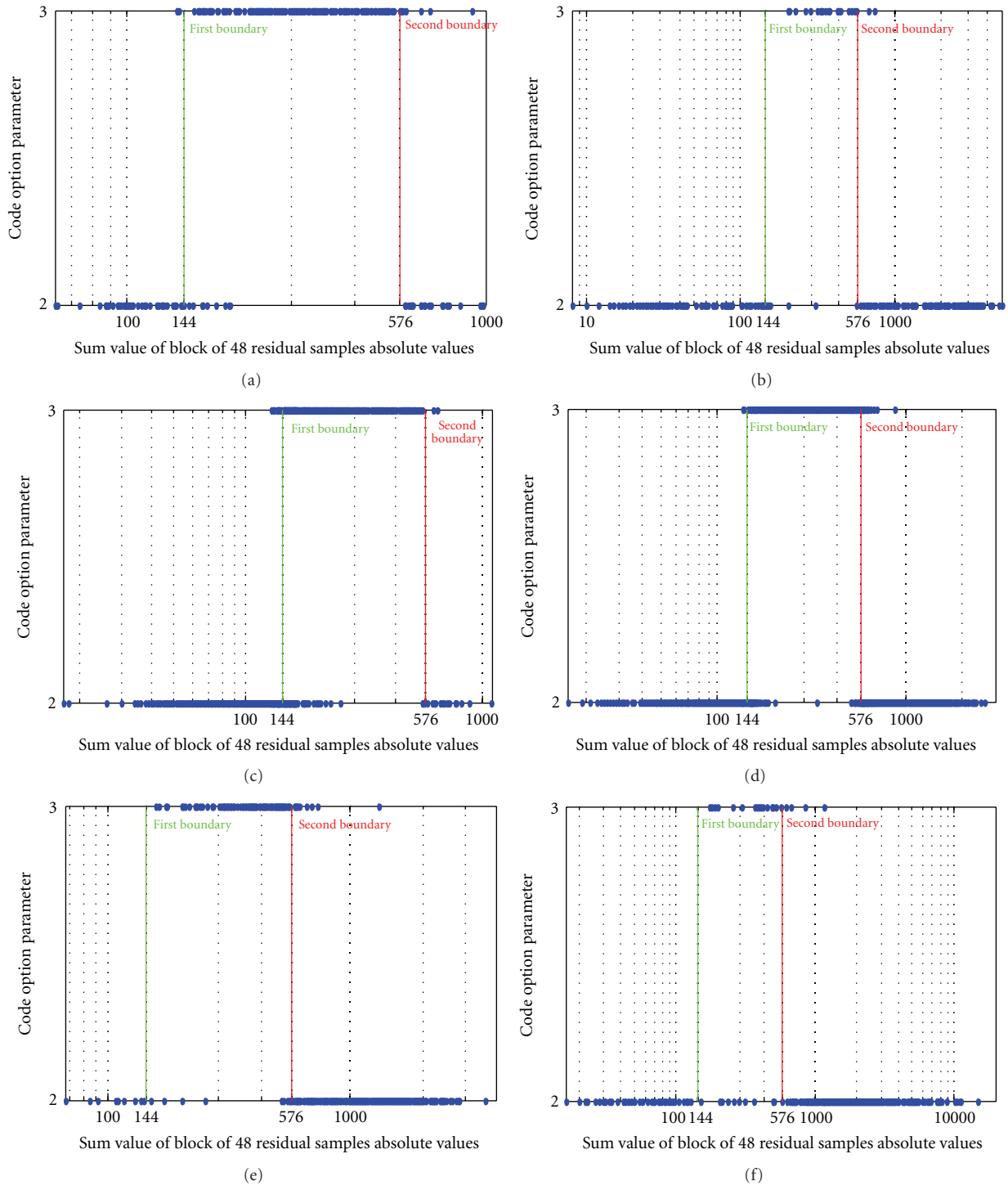

Figure 3: Code Option Parameter versus Sum value of Block of 48 residual samples absolute values for (a) FN101 temperature data set, (b) FN101 RH data set, (c) LU84 temperature data set, (d) LU84 RH data set, (e) LG20 temperature data set, and (f) LG20 RH data set.

where

$$b_i = \left\lceil \log_2(|d_i|) \right\rceil, \tag{2}$$

$$\text{Index} = \begin{cases} d_i & d_i \geq 0, \\ \left(2^{b_i} - 1\right) + d_i & d_i < 0. \end{cases} \tag{3}$$

Equation (3) returns the index position of each d_i within its group. $(\text{Index})|_{b_i}$ denotes the binary representation of Index over b_i bits. b_i is the category (group number) of d_i. It is also the number of lower order bits needed to encode the value of d_i. Note that if $d_i = 0$, l_i is not represented. Thus, at that instance, $c_i = h_i$. Once c_i is generated, it is appended

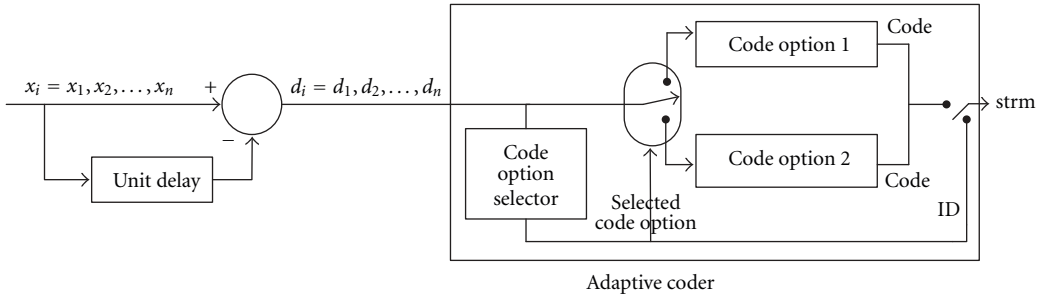

Figure 4: Functional block diagram of ALDC algorithm using the decision regions approach.

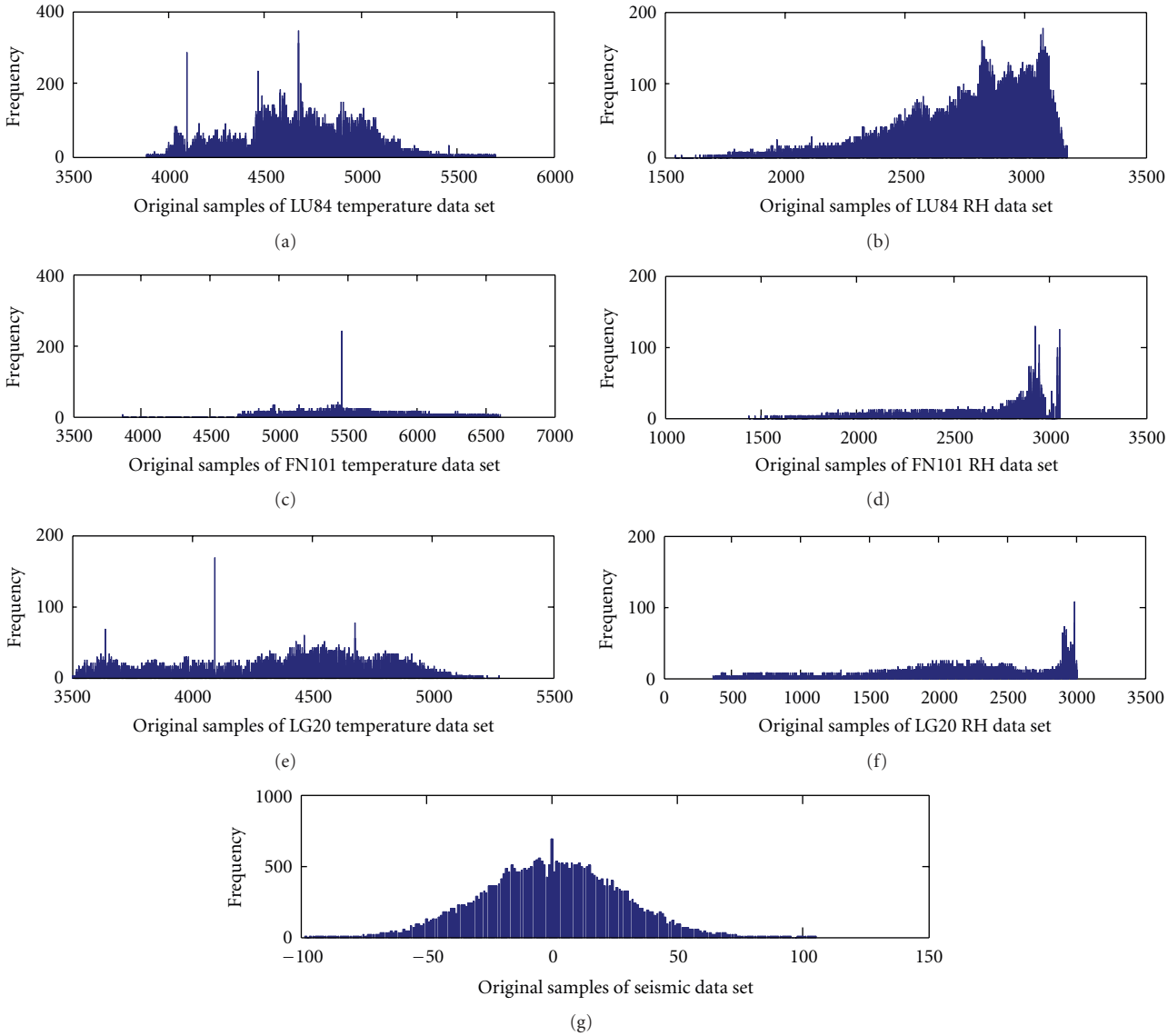

Figure 5: Frequency distribution plots of the raw test data sets: (a) LU84 temperature data set, (b) LU84 RH data set, (c) FN101 temperature data set, (d) FN101 RH data set, (e) LG20 temperature data set, (f) LG20 RH data set, and (g) Seismic data set.

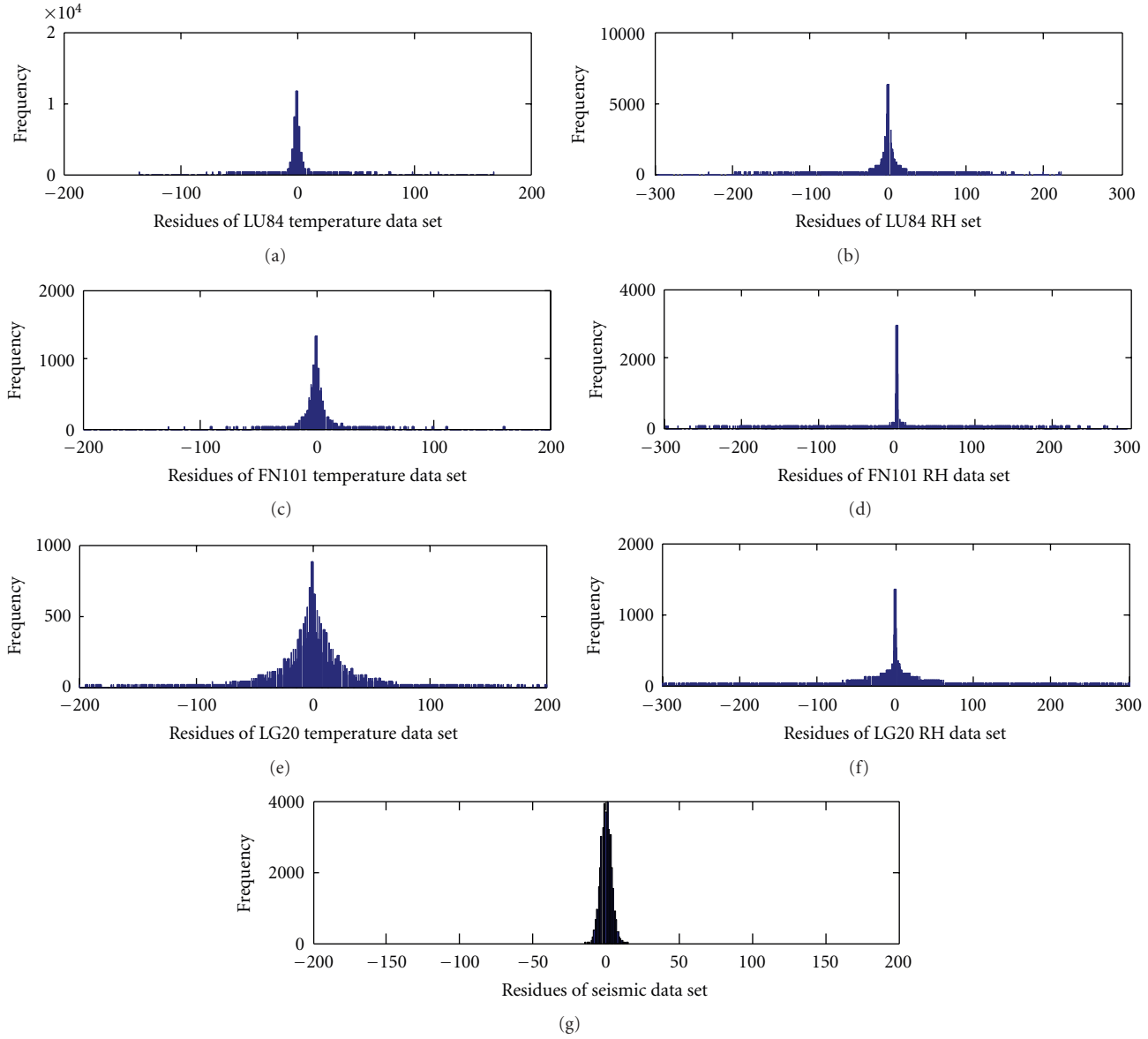

Figure 6: Frequency distribution plots of the residue of the test data sets: (a) LU84 temperature data set (b) LU84 RH data set (c) FN101 temperature data set (d) FN101 RH data set (e) LG20 temperature data set (f) LG20 RH data set (g) Seismic data set.

to the bit stream which forms the compressed version of the sequence of measures. Our proposed ALDC Scheme employs the principle of predictive coding to better capture the underlying temporal correlations that exist among sampled data in continuous monitoring. In predictive coding, a linear or nonlinear prediction models are used in the first stage, while a number of coding schemes are used in the second stage.

3.1. Prediction Model. The dynamic range of source symbols is a key factor in achieving compression. For this reason, we adopt a differential compression scheme to reduce the dynamic range of the source symbols thereby increasing its compressibility. The prediction approach adopted by us uses a linear model that is limited to taking the differences between consecutive sampled data. For our intended application which is the compression of environmental data such as temperature, relative humidity and seismic data, this prediction approach proves to be simple and efficient. In addition, this also ensures that the computational complexity of our compression scheme is as low as possible since sensor nodes in WSNs have relatively low computational power. Thus, the predicted sample $\hat{x}_i$ is given by

$$\hat{x}_i = x_{i-1}. \tag{4}$$

That is, the predicted sample is equal to the last observed sample. The residue (i.e., the error term) is then calculated

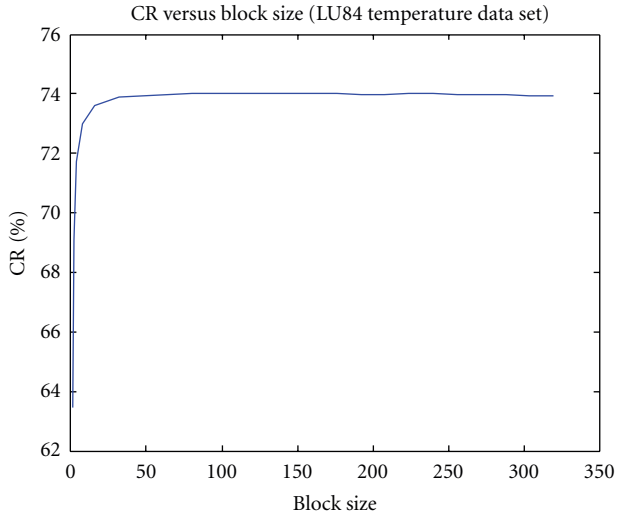

Figure 7: CR versus block size achieved by the ALDC algorithm for the LU84 temperature data set using the brute-force approach.

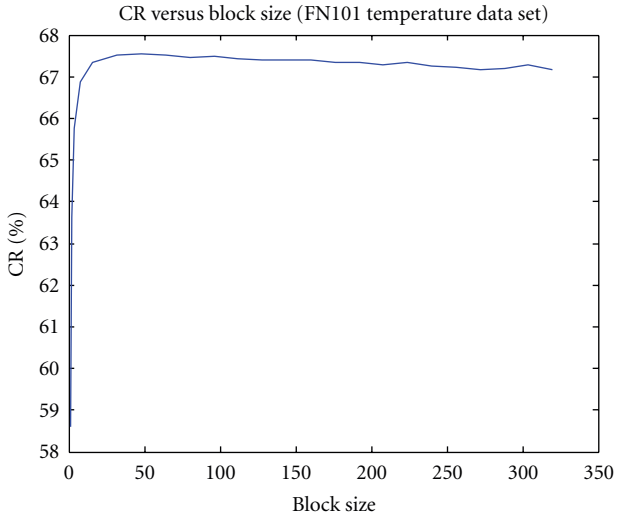

Figure 8: CR versus block size achieved by the ALDC algorithm for the FN101 temperature data set using the brute-force approach.

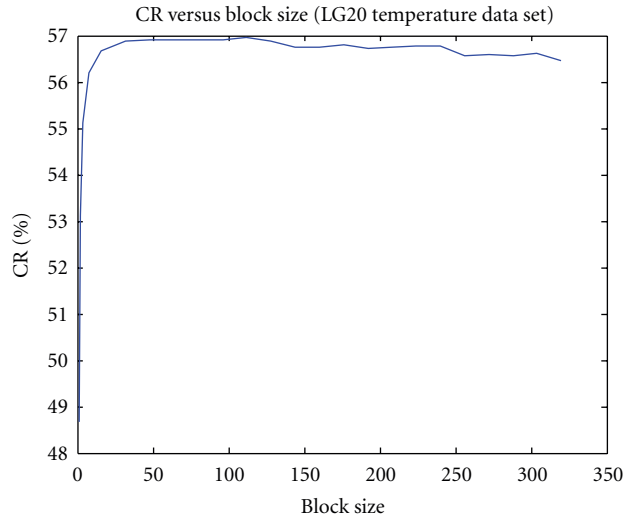

Figure 9: CR versus block size achieved by the ALDC algorithm for the LG20 temperature data set using the brute-force approach.

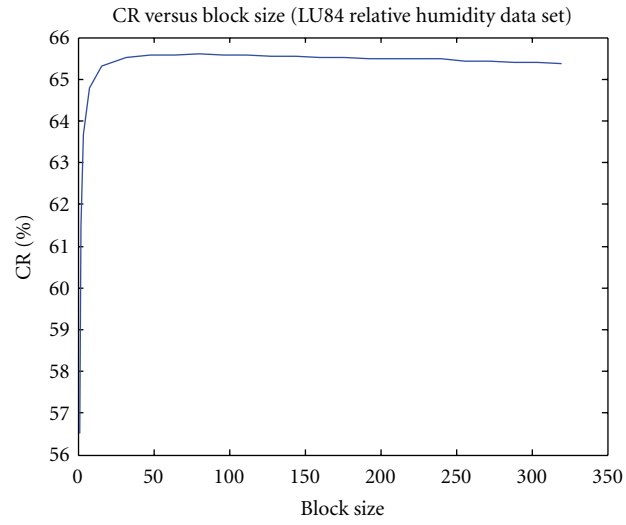

Figure 10: CR versus block size achieved by the ALDC algorithm for the LU84 relative humidity data set using the brute-force approach.

by subtracting the predicted sample from the current sample. Hence, the residue d_i is the difference

$$d_i = x_i - x_{i-1}. \tag{5}$$

In order to compute the first residue d_1 we assume that

$$x_0 = 2^{R-1}, \tag{6}$$

where R is the default measurement resolution of the incoming data set (i.e., R is the dynamic range of the source symbols under consideration). Note, each x_i is a positive-integer value in the range $[0, 2^R - 1]$ and it is represented in binary on R bits. We choose x_0 to be equal to the central positive-integer value among the 2^R possible positive-integer values. In our intended application using the test datasets, the default measurement resolution for the relative humidity and temperature datasets is 12 bits and 14 bits, respectively. Therefore, for each application, x_0 is known by both the encoder and decoder. Consequently, the algorithm is adaptable to different data sources since x_0 is related to the incoming data. The computed residue d_i is then used as input to the entropy encoder. That is, d_i is used to losslessly encode x_i using the coding schemes described in Section 3.2.

3.2. Entropy Coding. In order to achieve maximal compression ratio and by extension maximal energy saving, we propose to implement ALDC algorithm that compresses blocks of sampled data at a time using two ALEC code options adaptively. Our proposed ALDC algorithm operates in one pass and can be applied to different data types. Our

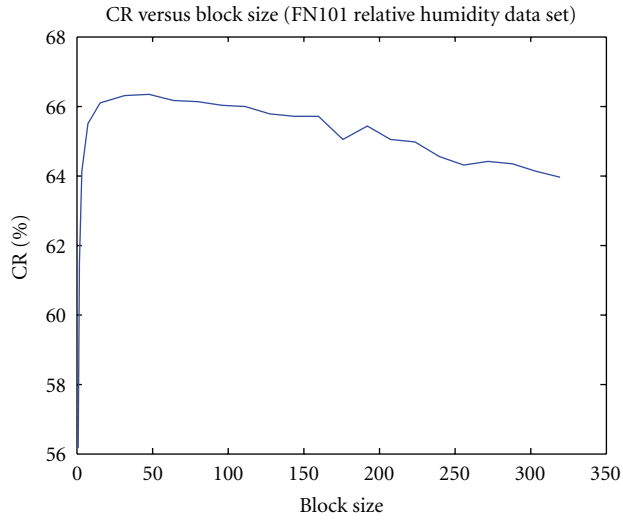

Figure 11: CR versus block size achieved by the ALDC algorithm for the FN101 relative humidity data set using the brute-force approach.

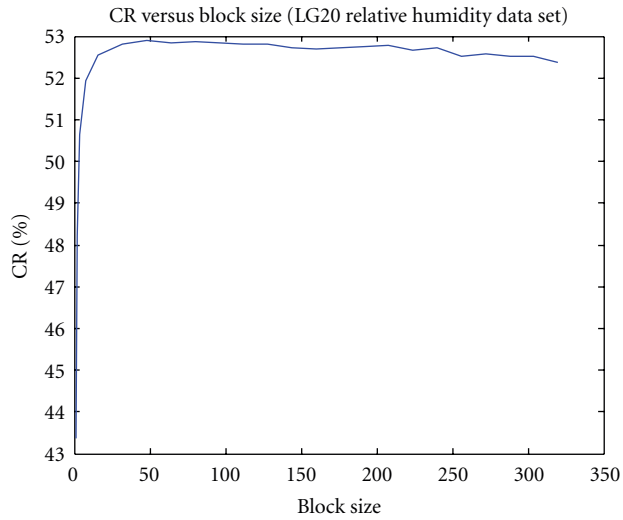

Figure 12: CR versus block size achieved by the ALDC algorithm for the LG20 relative humidity data set using the brute-force approach.

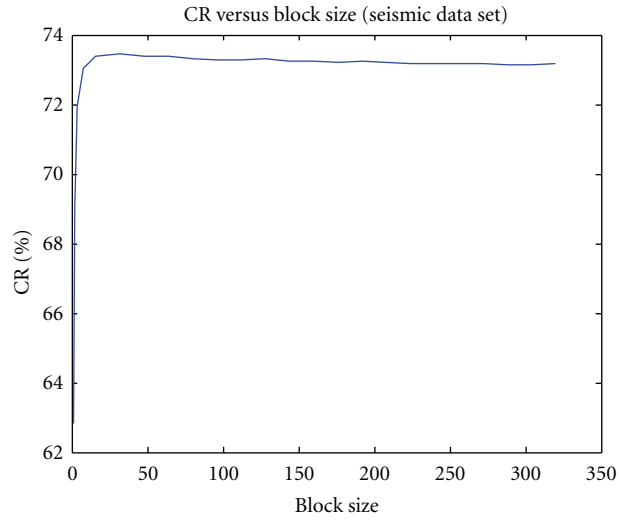

Figure 13: CR versus block size achieved by the ALDC algorithm for the Seismic data set using the brute-force approach.

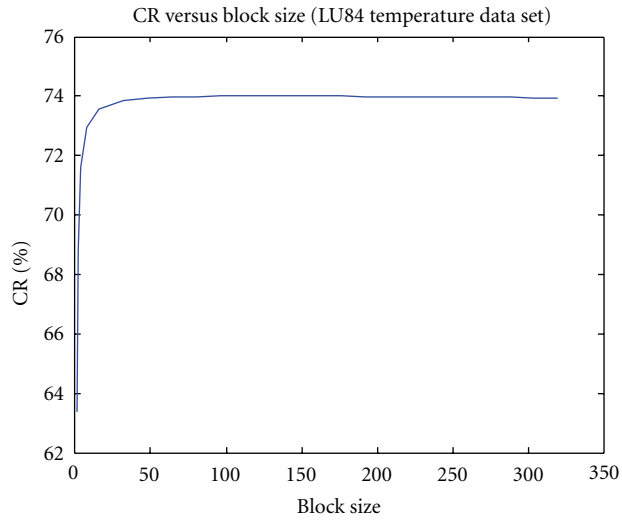

Figure 14: CR versus block size achieved by the ALDC algorithm for the LU84 temperature data set using the decision regions approach.

entropy coding problem is how to efficiently encode block of n integer-valued samples at a time using two ALEC code options adaptively. Two different approaches to solve this entropy coding problem will be discussed in this section. The approaches are, namely, the brute-force approach and the decision regions approach.

3.2.1. The Brute-Force Approach. Figure 1 shows the functional block diagram of the implementation of the ALDC algorithm using the brute-force approach. Code option 1 and code option 2 represents 2-Huffman Table ALEC and 3-Huffman Table ALEC, respectively. From Figure 1, the block of n-samples x_i is preprocessed by the simple unit-delay predictor to obtain block of n-residues d_i. d_i is then encoded (compressed) by the adaptive coder using both code options. The sizes of the encoded bitstreams generated by using the two code options are then compared. The code option that yields the smallest encoded bitstream size (i.e., highest compression) is then selected. The encoded bitstream generated by this code option is then appended to the code option identifier (ID) and thereafter sent to the sink. The decoder uses the ID to identify the code option used in encoding d_i. Repeat the procedure until the end of the source data is reached. The pseudocode of the compress function using the brute-force approach is given in Algorithm 4. The brute-force approach guarantees that optimal compression ratio is attained for each data set since it is always the best code option that is selected for each block of n samples. However, the brute-force approach requires more memory

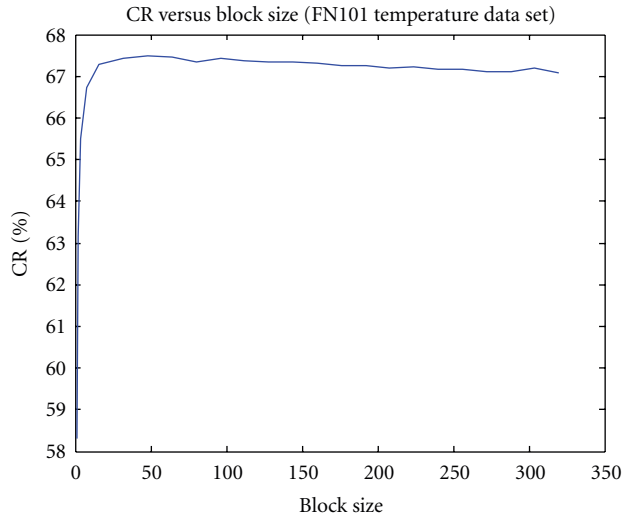

Figure 15: CR versus block size achieved by the ALDC algorithm for the FN101 temperature data set using the decision regions approach.

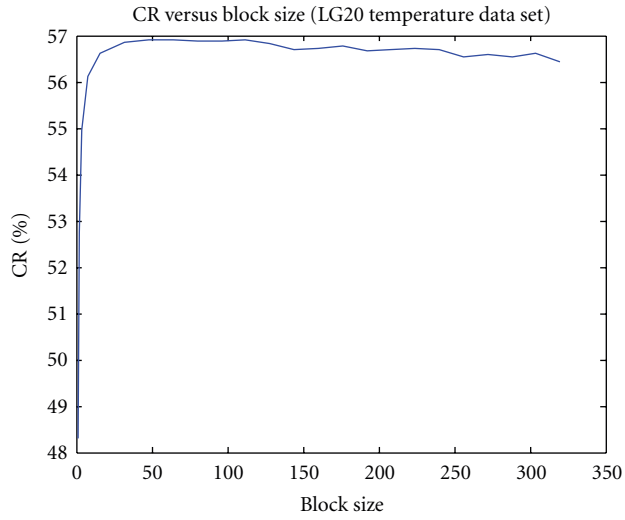

Figure 16: CR versus block size achieved by the ALDC algorithm for the LG20 temperature data set using the decision regions approach.

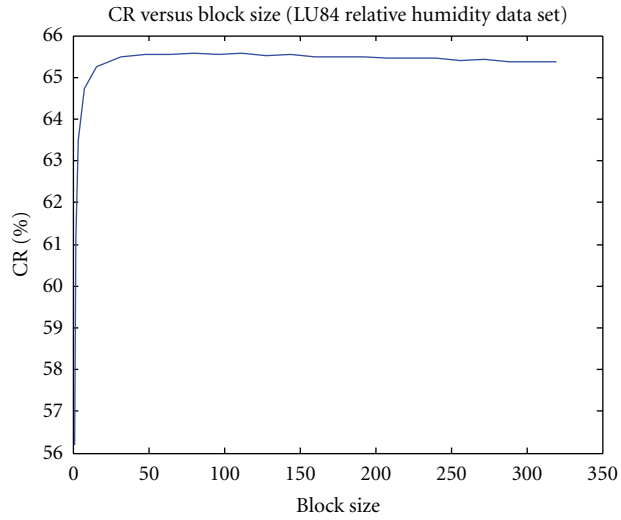

Figure 17: CR versus block size achieved by the ALDC algorithm for the LU84 relative humidity data set using the decision regions approach.

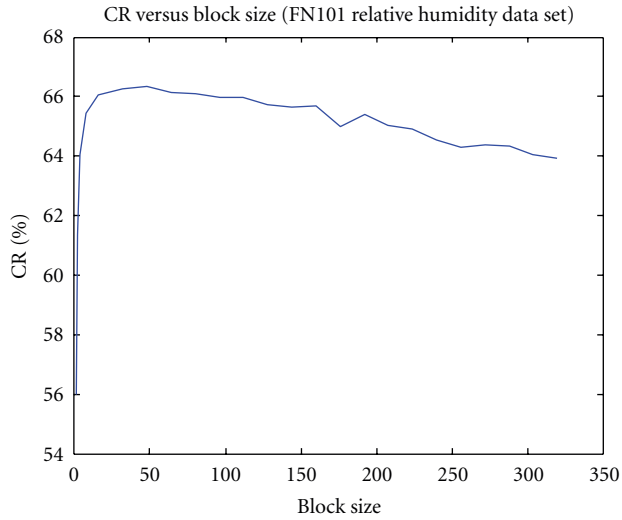

Figure 18: CR versus block size achieved by the ALDC algorithm for the FN101 relative humidity data set using the decision regions approach.

(for buffering the encoded bitstreams of both code options for comparison) and it is also computational intensive (since encoding is done by both code options for each block of n-samples d_i).

3.2.2. The Decision Regions Approach. As stated in Section 3.2.1, the brute-force approach requires more memory (for buffering the encoded bitstreams of both code options for comparison) and it is also computational intensive (since encoding is done by both code options for each block of n-samples d_i). However, the sensor node has stringent constraint in terms of memory, computational power and energy. We therefore turn our attention to the issue of selecting a code option that efficiently encode block of n-samples d_i without using the brute-force approach since it has been shown that the approach is unnecessarily complex. In [28], the authors introduced a high performance adaptive coding module using the brute-force approach in the selection of a code option. Because the brute-force approach is computationally exhaustive and/or hardware demanding, the authors later proposed a simpler alternative to the brute-force approach that uses a table of decision regions that is solely based on the length of the fundamental sequence of n-standard source samples (nonnegative integers). The length of the fundamental sequence of n-standard source samples is essentially the sum of the n-standard source samples in a block plus n (the block size). The calculated sum is then used alongside the table of

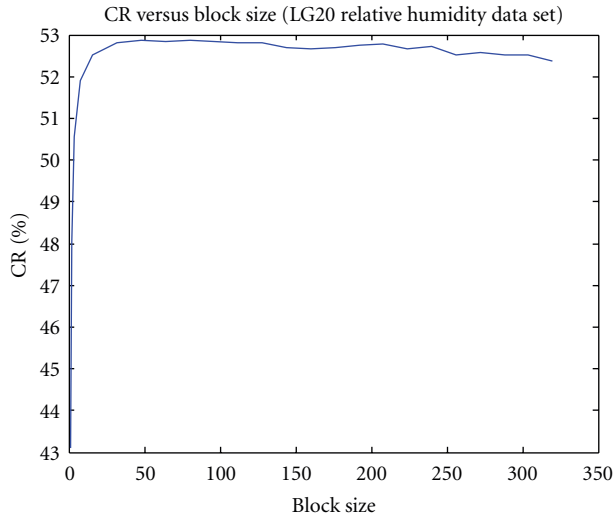

Figure 19: CR versus block size achieved by the ALDC algorithm for the LG20 relative humidity data set using the decision regions approach.

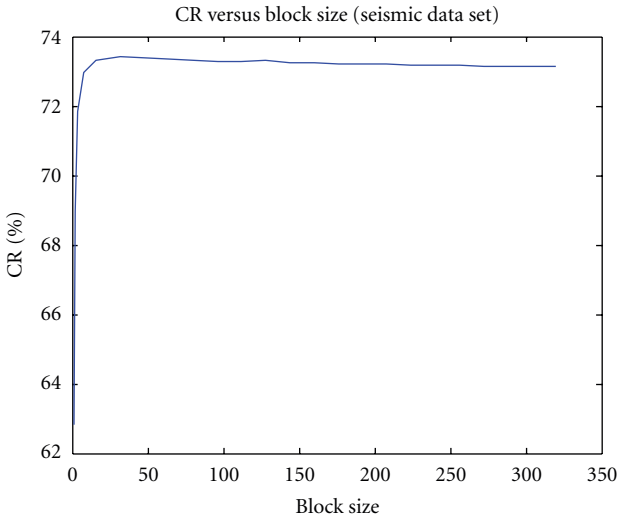

Figure 20: CR versus block size achieved by the ALDC algorithm for the Seismic data set using the decision regions approach.

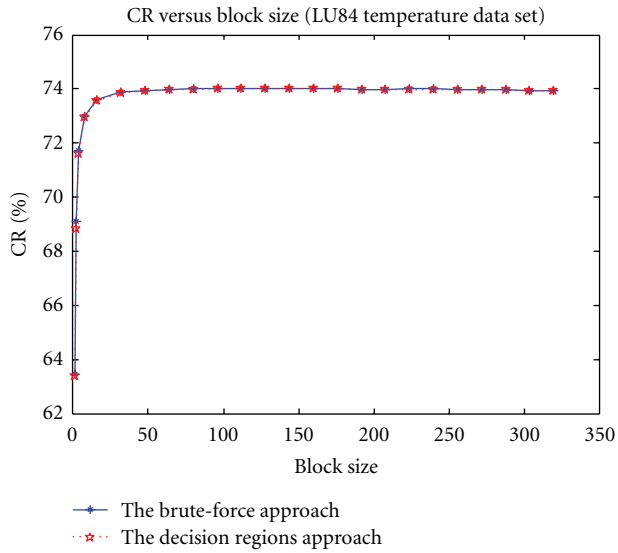

Figure 21: Performance comparison between the brute-force approach and the decision regions approach of ALDC for the Lu84 temperature data set.

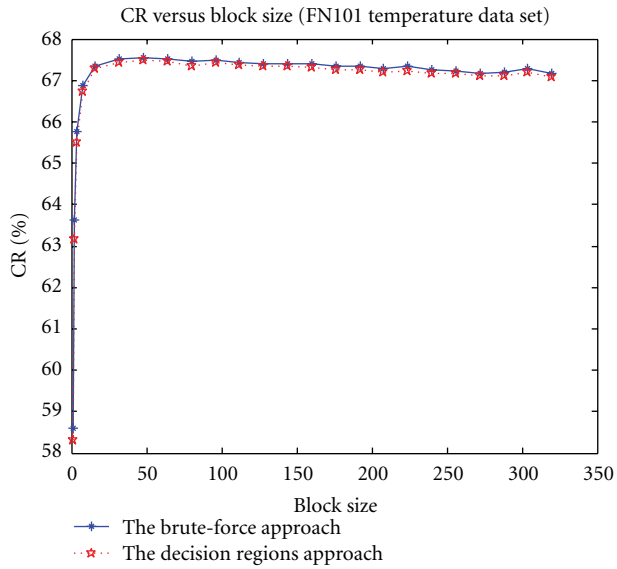

Figure 22: Performance comparison between the brute-force approach and the decision regions approach of ALDC for the FN101 temperature data set.

decision regions to select the best code option for encoding. Thus, under this new approach, only one code option was used per block of n-standard source samples. This led to a lot of savings in times of both computational requirements and hardware. Motivated by the simplicity of this decision regions approach, we set out to find out if for our proposed ALDC we can use similar decision regions approach defined solely by certain sum expression for best code option selection using empirical method.

To this end, using the brute-force approach discussed in Section 3.2.1 alongside its pseudocode in Algorithm 4, we generate the pattern of code options usage while compressing each data set by repeating the following procedures for each block of n-residues d_i until the end of the data set is reached:

(a) Compute the sum of the absolute value of each residual sample in a block of n-residues d_i. Store the computed sum in the SUM array.

(b) Encode (compress) the block of n-residues d_i using both code options (namely, 2-Huffman Table ALEC and 3-Huffman Table ALEC).

(c) Select the best code option that yields the smallest encoded bitstream size for each block of n-residues d_i. If 2-Huffman Table ALEC is the best code option selected, then the code option identifier (ID) of 2 is generated and stored in the ID array, otherwise the

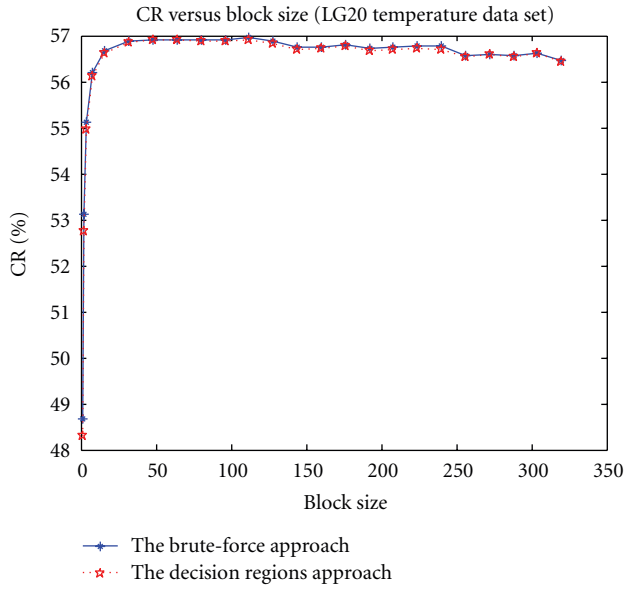

Figure 23: Performance comparison between the brute-force approach and the decision regions approach of ALDC for the LG20 temperature data set.

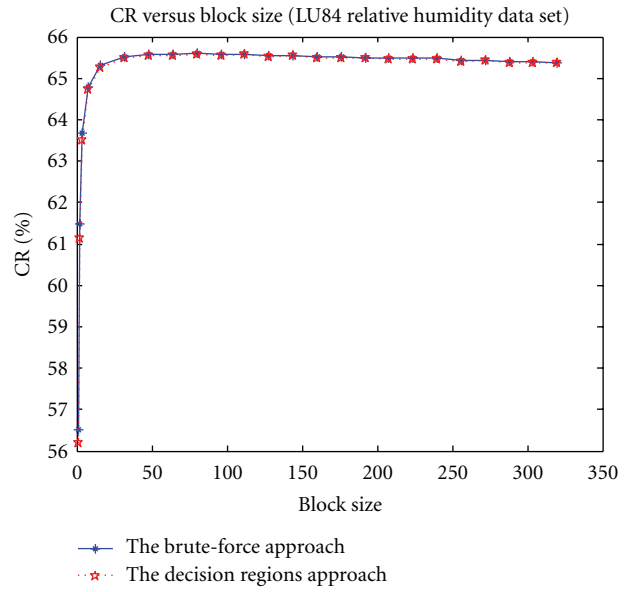

Figure 24: Performance comparison between the brute-force approach and the decision regions approach of ALDC for the Lu84 relative humidity data set.

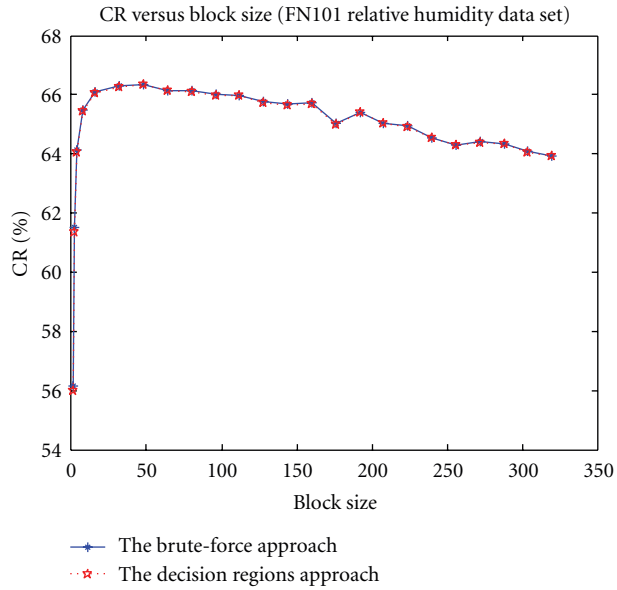

Figure 25: Performance comparison between the brute-force approach and the decision regions approach of ALDC for the FN101 relative humidity data set.

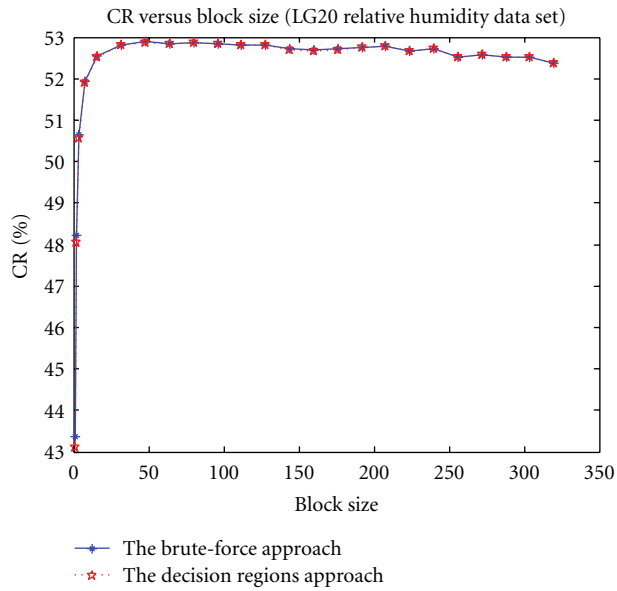

Figure 26: Performance comparison between the brute-force approach and the decision regions approach of ALDC for the LG20 relative humidity data set.

code option identifier (ID) of 3 is generated instead and stored in ID array.

Procedures (a) to (c) are repeated for block sizes of 32 and 48 for different data sets. Thereafter, we plotted the ID arrays against the corresponding SUM arrays using data markers only for the different test data sets. These plots are given in Figures 2 and 3 for block size of 32 and 48 samples, respectively. Note that, some of the data points in the plots are plotted several times. As seen from the plots (Figures 2 and 3), the compression performance of the two code options overlaps at two regions. For the block size of 32, the two regions are the sum values in the range $[80, 112]$ and $[368, 400]$. Similarly, for block size of 48, the two regions within which the performance of the two code options overlaps are the sum values in the range $[120, 168]$ and $[552, 600]$. Depending on the block size (32 or 48), any sum value in these two regions can be used as decision regions boundary sum value resulting in approximately the same

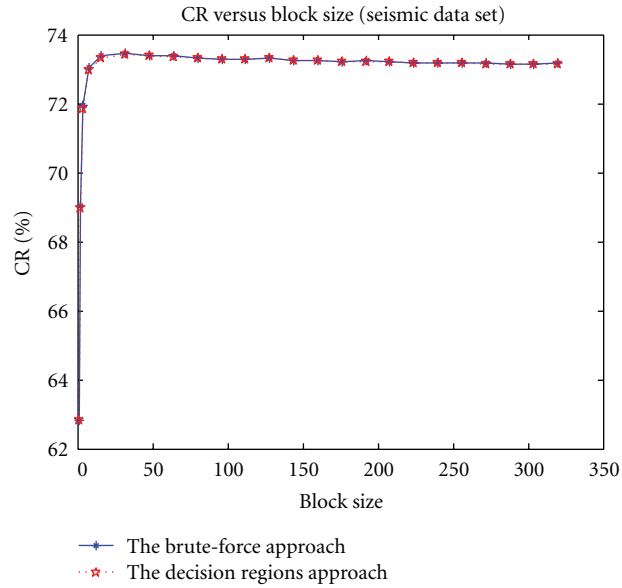

FIGURE 27: Performance comparison between the brute-force approach and the decision regions approach of ALDC for the Seismic data set.

compression ratio. For simplicity, we define the decision regions boundary sum value as that sum value in each range that is a multiple of block size. Thus, for the block size of 32, with [80, 112] and [368, 400] as the two overlapping regions, the decision regions boundary sum values are taken to be 96 (3×32) and 384 (12×32), respectively. Similarly, for the block size of 48 in which the overlapping regions are [120, 168] and [552, 600], the decision regions boundary sum values are taken to be 144 (3 × 48) and 576 (12 × 48), respectively. We conclude therefore from the foregoing that given block size n, the two decision regions boundary sum values can be computed as $3n$ and $12n$, respectively, thereby making the two decision regions boundary sum values multiples of n (block size). The two decision regions boundary sum values are indicated in the plots (Figures 2 and 3) as "First boundary" and "second boundary" respectively. Table 4 gives the summary of the decision regions used by our proposed ALDC algorithm. Using Table 4, the encoding procedure using the decision regions approach then simplifies to the following steps:

(1) Compute the sum

$$F = \sum_{i=1}^{n} |d_i|. \tag{7}$$

(2) Check if $F \leq 3n$. If this condition is satisfied, then 2-Huffman Table ALEC code option is selected and its code option identifier ID is generated. The encoded bitsream from the 2-Huffman Table ALEC is then concatenated to ID. Otherwise, move to the next step.

(3) Check if $3n < F \leq 12n$. If this condition is satisfied, then 3-Huffman Table ALEC code option is selected and its code option identifier ID is generated. The encoded bitsream from the 3-Huffman Table ALEC is then concatenated to ID. Otherwise, move to the next step.

(4) Check if $12n < F$. If this condition is satisfied, then 2-Huffman Table ALEC code option is selected and its code option identifier ID is generated. The encoded bitsream from the 2-Huffman Table ALEC is then concatenated to ID.

The functional block diagram of the implementation of the ALDC algorithm using the decision regions approach is given in Figure 4. As will be seen in Section 4, the compression performance of the ALDC decision regions approach is almost the same with those obtained using the brute-force approach. This shows the correctness of the decision regions in Table 4 that were arrived at through empirical observations. Thus, we recommend that users of the ALDC compression scheme should use the decision regions approach. The performance of the ALDC brute-force approach only serves as benchmark to users of the ALDC compression scheme.

3.3. Numerical Example Using the Decision Region Approach. In this section, we present a numerical example to show the steps of the ALDC algorithm using the decision regions approach. Suppose a block of incoming temperature samples with ADC resolution of 14 is: $x_i = \{x_1, x_2, x_3, \ldots, x_n\} = \{8202, 8202, 8202, 8201, 8202, 8202, 8202, 8208\}$

(1) We compute the residues d_i by applying (5) and (6). Thus, we have

$$d_i = \{d_1, d_2, d_3, \ldots, d_n\} = \{10, 0, 0, -1, 1, 0, 0, 6\}. \tag{8}$$

(2) Applying (7), we compute the sum of the absolute value of the residues in the block of 8-residues d_i. That is, we compute

$$F = \sum_{i=1}^{8} |d_i| = 10 + 0 + 0 + 1 + 1 + 0 + 0 + 6 = 18. \tag{9}$$

(3) Next, we determine the boundaries that define the decision regions in Table 4:

The first boundary $= 3n = 3 \times 8 = 24$,
The second boundary$= 12n = 12 \times 8 = 96$.

(4) Next, we determine the F region using Table 4. Since $F = 18 < 24$, that means F falls within the first decision region. Thus, the 2-Huffman Table ALEC is selected as the best code option for encoding d_i and its code option identifier ID is generated. Note that, since we are only using two code options, the code option identifier ID is either "0" (for 2-Huffman Table ALEC) or "1" (for 3-Huffman Table ALEC).

(5) Next, we encode d_i using Algorithm 1 and append the encoded bitstream to the ID generated in step 4 above. The final output of the encoded values is:

```
2TableALECencoder(d_i,n, code)
// encode() is the encode function
// d_i is the current residue value
// n is the block size (the number of residue values to be encoded at a time)
// code is the encoded bitstream of n d_i
// *denotes concatenation

// encode block of n d_i using the first Huffman Table of the 2-Huffman Table ALEC Coder
CALL encode() with block of n d_i and Table A RETURNING c_i
SET c_iA To c_i
// compute the size of the encoded bitstream c_iA
SET size_A TO length(c_iA)
// encode the same block of n d_i using the second Huffman Table of the 2-Huffman Table ALEC Coder
CALL encode() with block of n d_i and Table B RETURNING c_i
SET c_iB To c_i
// compute the size of the encoded bitstream c_iB
SET size_B TO length(c_iB)
// compare size_A and size_B and select the encoded bitstream with the least compressed size
IF size_A <= size_B THEN
   // generate the table identifier of Table A
   SET ID TO "0"
   // append encoded bitstream c_iA to ID
   SET code TO ID * c_iA
ELSE
   // generate the table identifier of Table B
   SET ID TO "1"
   // append encoded bitstream c_iB to ID
   SET code TO ID * c_iB
ENDIF
RETURN code
```

Algorithm 1: Pseudo-code of 2-Huffman Table ALEC.

"0 0 1001 1010 00 00 01 0 01 1 00 00 101 110"

The output is colour coded and separated from each other for explanation purpose. The red "0" is a code identifier ID that tells the decoder that it was 2-Huffman Table ALEC code option that was used to encode the block of 8 samples. The green "0" is a table identifier ID that tells the decoder that it was Huffman Coding Table A given in Table 1 that was used by the 2-Huffman Table ALEC code option for encoding the block of 8 samples. Thus, encoding is done in accordance with Algorithm 3 using Table 1. Note that, the encoding function in Algorithm 3 encodes each sample as two parts: Huffman code for the group and binary code representation of the index position of each d_i in the group using b_i bits. If d_i is zero, then b_i is also zero and at that instance, the binary code representation is not required. Thus, the blue "1001" and the pink "1010" represent the group and binary code of residual value 10, respectively. Next, the two blue "00" represent the group code of two residual values 0 since the binary code representation is not required. Following next is the blue "01" and pink "0" that, respectively, represent the group and binary code of residual value −1. Next, is the blue "01" and pink "1" that, respectively, represent the group and binary code of residual value 1. Next, is the two blue "00" that represents the group code of two residual values 0. Finally, the blue "101" and the pink "110" represent the group and the binary code representation of residual value 6. Putting all the codes together, the final output of the encoded values that are sent to the decoder is 00100110100000010011000010111 0 which is made up of a total of 30 bits against 112 bits if the original sample values were to be transmitted uncompressed. Thus, for the given block of 8 incoming temperature samples, the total savings in terms of bits is 82 bits. This translates to energy savings for the sensor node since it has to now transmit fewer numbers of bits over its radio.

4. Simulations and Analysis

To verify the effectiveness of our proposed algorithm, we tested it against various real-world environmental data sets discussed in Section 4.1. We considered relative humidity data sets, temperature data sets, and seismic data set. The compression performance was calculated in terms of compression ratio, computed by using the following formula:

$$\mathrm{CR} = 100 \times \left(1 - \frac{\mathrm{comp}}{\mathrm{orig}}\right)\%, \tag{10}$$

where comp is the number of bits obtained after compression and orig is the uncompressed data size. Each uncompressed sample data is represented by 16-bit unsigned integers.

```
3TableALECencoder(d_i,n, code)
// encode() is the encode function
// d_i is the current residue value
// n is the block size (the number of residue values to be encoded at a time)
// code is the encoded bitstream of n d_i
// *denotes concatenation

// encode block of n d_i using the first Huffman Table of the 3-Huffman Table ALEC Coder
CALL encode() with block of n d_i and Table A RETURNING c_i
SET c_iA To c_i
// compute the size of the encoded bitstream c_iA
SET size_A TO length(c_iA)
// encode the same block of n d_i using the second Huffman Table of the 3-Huffman Table ALEC Coder
CALL encode() with block of n d_i and Table B RETURNING c_i
SET c_iB To c_i
// compute the size of the encoded bitstream c_iB
SET size_B TO length(c_iB)
// encode the same block of n d_i using the third Huffman Table of the 3-Huffman Table ALEC Coder
CALL encode() with block of n d_i and Table C RETURNING c_i
SET c_iC To c_i
// compute the size of the encoded bitstream c_iC
SET size_C TO length(c_iC)
// compare size_A, size_B and size_C and select the encoded bitstream with the least compressed size
IF size_A <= min(size_B, size_C) THEN
   // generate the table identifier of Table A
   SET ID TO "10"
   // append encoded bitstream c_iA to ID
   SET code TO ID * c_iA
ELSEIF size_B <= min(size_A, size_C) THEN
   // generate the table identifier of Table B
   SET ID TO "11"
   // append encoded bitstream c_iB to ID
   SET code TO ID * c_iB
ELSEIF size_C <= min(size_A, size_B) THEN
   // generate the table identifier of Table C
   SET ID TO "0"
   // append encoded bitstream c_iC to ID
   SET code TO ID * c_iC
ENDIF
RETURN code
```

Algorithm 2: Pseudo-code of 3-Huffman Table ALEC.

4.1. Data Sets. Real-world environmental monitoring WSN datasets from SensorScope [29] were used in our simulations. We used relative humidity and temperature measurements from three SensorScope deployments: Le Gènèpi Deployment, HES-SO FishNet Deployment and LUCE Deployment. Publicly accessible data sets were used to make the comparison as fair as possible. These deployments use a TinyNode node [30] which comprises of a TI MSP430 microcontroller, a Xemics XE120,5 radio and a Sensirion SHT75 sensor module [31]. Both the relative humidity and temperature sensors are connected to a 14-bit analog-to-digital converter (ADC). The default measurement resolution for raw relative humidity (*raw_h*) and raw temperature (*raw_t*) is 12 bits and 14 bits respectively. Each ADC output *raw_h* and *raw_t* are converted into measure h and t in percentage and degree Celsius respectively as described in [31]. The data sets that are published on SensorScope deployments correspond to physical measures h and t. But the compression algorithms work on *raw_h* and *raw_t*. Therefore, before applying the compression algorithm, the physical measures h and t are converted to *raw_h* and *raw_t* by using the inverted versions of the conversion functions in [31]. Table 5 summarizes the main characteristics of the datasets. See [3] for further details regarding the characteristics of these data sets. In addition, we also used a seismic data set collected by the OhioSeis Digital Seismographic Station located in Bowling Green, Ohio, for the time interval of 2:00 PM to 3:00 PM on 21 September 1999 (UT) [32]. We compute the information entropy $H = -\sum_{i=1}^{N} p(x_i) \cdot \log_2 p(x_i)$ of the original data sets, where N is the number of possible values of x_i (the output of the ADC) and $p(x_i)$ is the probability mass function of x_i. In addition, the information entropy $H_d = -\sum_{i=1}^{N} p(d_i) \cdot \log_2 p(d_i)$ of the residual signal was also computed. These are all recorded in Table 6.

```
encode(d_i, TABLE, c_i)
// d_i is the current residue value
// TABLE is the variable length Huffman codes used in encoding
// b_i is the category (group number) of d_i
// b_i is also the number of lower order bits needed to encode the value of d_i
// c_i is the encoded bitstream of d_i
// h_i is the variable-length Huffman code that codifies the category (group) of d_i
// l_i is the variable-length integer code that codifies the index position of d_i
// within its group (category)
// *denotes concatenation
// (Index)| b_i denotes the binary representation of index over b_i bits

// compute d_i category
IF d_i = 0 THEN
   SET b_i TO 0
ELSE
   SET b_i TO ⌈log_2 (|d_i|)⌉
ENDIF
// extract h_i the variable length Huffman code from TABLE
SET h_i TO TABLE [b_i]
// build c_i
IF b_i = 0 THEN
   // l_i is not needed
   SET c_i TO h_i
ELSE
   // build l_i
   SET l_i TO (Index)| b_i
   // build c_i
   SET c_i TO h_i * l_i
ENDIF
RETURN c_i
```

Algorithm 3: Pseudo-code of the encode () function.

Figure 5 shows the distribution plots of the raw test data sets and Figure 6 shows the distribution of differences between consecutive sample data (residue) of the test data sets. While there are differences in distributions in raw data as seen in Figure 5, the residual distributions as seen in Figure 6 are similar. Whenever the residues of any data sets have lower mean and lower standard deviation, their entropy will be low. Hence, if entropy compression algorithms (like our proposed scheme) are applied to a low entropy data set, the compression ratio achievable will be high. Our proposed ALDC scheme operates only on residues. Thus, since the residual distributions of many real-world continuous monitoring data sets are similar, our proposed ALDC (using either the Brute-Force Approach or the Decision Regions Approach) algorithm can be applied to different types of data sets and still yields satisfactory compression ratios." This is as a result of the adaptive use of different Huffman coding tables that handles different levels of data correlation (entropy) by the two code options.

4.2. Compression Performance. The compression performance of our proposed ALDC algorithm will be computed using (10) for different values of n (block size) and for the seven real-world data sets discussed in Section 4.1. In Section 3.2, we presented two different approaches for the implementation of the ALDC algorithm. These approaches are the brute-force approach and the decision regions approach. The performance of our proposed ALDC algorithm will be evaluated for each of these two different approaches. For our simulations, n (block size) takes the value 1, 2, 4, 8, 16, 32, 48, 64, 80, 96, ... 320.

4.2.1. Compression Performance Using the Brute-Force Approach. For each data set, using the brute-force approach, the compression performance of ALDC is computed for different value of n. Figures 7, 8, 9, 10, 11, 12, and 13 shows the compression ratio versus block size achieved by the ALDC algorithm for the seven real-world data sets using the brute-force approach. As evident from Figures 7 to 13, the compression performance of ALDC algorithm for each of the seven data sets using the brute-force approach increases with respect to the increase in the block size. For very small values of n, lower compression performances were obtained due to high ID overhead cost incurred that overweigh the compression benefits. However, the compression performances obtained were significantly higher for block sizes in the range $4 \leq n \leq 48$. This is due to low ID overhead cost together with high adaptability to changes in the sensed data statistics for such block sizes. Values of n (block size) beyond 48 results in less improvement to the compression

```
BruteForceCompress(x_i,x_{i-1},n, y )
// x_i is the current sensor reading(s)
// x_i_1 is the immediate past sensor reading(s)
// n is the block size (the number of samples read each time)
// y is the final encoded bitstream
// 2TableALECencoder() is the 2-Huffman Table ALEC encode function
// 3TableALECencoder() is the 3-Huffman Table ALEC encode function

// compute the residue d_i
SET d_i TO x_i − x_{i-1}
// encode the residue d_i
// encode block of n d_i using the 2-Huffman Table ALEC encode function
CALL 2TablesALECencoder() with block of n d_i RETURNING code
SET codeA To code
// compute the size of the encoded bitstream codeA
SET size_A TO length(codeA)
// encode the same block of n d_i using the 3-Huffman Table ALEC encode function
CALL 3TablesALECencoder() with the same block of n d_i RETURNING code
SET codeB To code
// compute the size of the encoded bitstream codeB
SET size_B TO length(codeB)
// compare size_A and size_B and select the encoded bitstream with the least compressed size
IF size_A <= size_B THEN
   // generate the code option identifier for the 2-Huffman Table ALEC encoder
   SET ID TO "0"
   // append encoded bitstream codeA to ID
   SET strm TO ID ∗ codeA
ELSE
   // generate the code option identifier for the 3-Huffman Table ALEC encoder
   SET ID TO "1"
   // append encoded bitstream codeB to ID
   SET strm TO ID ∗ codeB
ENDIF
// append bitstream strm to y
SET y TO y ∗ strm
RETURN y
```

Algorithm 4: Pseudo-code of the compress function of the ALDC using the brute-force approach.

ratio and the compression performance even degrades after a certain point as can be seen in some of the plots. Thus, for optimum compression performance using the ALDC algorithm, the value of n (block size) could be fixed at 48 ($n = 48$).

4.2.2. Compression Performance Using the Decision Regions Approach. The compression performance of our proposed ALDC is computed for different value of n and for each of the seven data sets using the decision regions approach. The results are plotted in Figures 14, 15, 16, 17, 18, 19, and 20. Figures 14 to 20 has the same features and characteristics with the corresponding plots in Figures 7 to 13. Note that, the range of the compression ratio achievable by ALDC for each of the seven data sets is a function of the entropy H_d (see Table 6 for the entropy of the residual data sets) of the residual signal fed into the encoder. Thus, data sets with low entropy (e.g., LU84 temperature data set) yields high compression ratio, while data sets with high entropy (e.g., LG20 relative humidity data set) yields low compression ratio. Thus, the data feature that affects more the compression performance of our proposed algorithm is the entropy H_d of the residual signal fed into the encoder. This confirms that ALDC is actually an entropy encoder.

4.2.3. Performance Comparison between the Brute-Force Approach and the Decision Regions Approach. To ascertain the correctness and effectiveness of our proposed decision region approach method of implementing ALDC, we in this section compare its compression performances (Figure 14 to Figure 20) with those (Figure 7 to Figure 13) obtained using the brute-force approach. For ease of comparison, we plotted the corresponding figures of all the seven data sets on the same plot. These plots are shown in Figure 21, 22, 23, 24, 25, 26, and 27. From the plots (Figure 21 to Figure 27), it can be seen that the compression ratio performance achieved by the decision regions approach of ALDC algorithm for the seven data sets in use is almost the same and for some data sets same with those obtained using the brute-force approach. The slight difference noticed at some points is due

to the overlapping performance of the two code options at and around the two boundary regions. Overall, we can say that the performance of the decision regions approach is equivalent to that obtained using the brute-force approach. The decision regions approach is computationally more lightweight than the brute-force approach and it require less resources which makes it suitable for implementation in WSNs. In view of this, we recommend to every user of our proposed ALDC algorithm to implement only the decision regions approach. Henceforth, any mention of the ALDC algorithm in this article should be taken to mean its decision regions approach.

4.3. Performance Comparison with Other Lossless Compression Schemes. In this section, we present our simulation results that demonstrate the lossless compression performance and effectiveness of our proposed ALDC algorithm. The lossless compression performance of our proposed ALDC algorithm using the decision regions approach (with block size of 32 and 48) and that of other recently proposed lossless compression algorithms like LEC and S-LZW are given in Table 7 for all the seven real-world data sets. The compression performance that was achieved by the S-LZW algorithm was adopted from [3] with the following fixed parameters: MINI-CACHE ENTRIES = 32, MAX DICT ENTRIES = 512, BLOCK SIZE = 528 bytes, and DICTIONARY STRATEGY = Frozen [3, 15]. In addition, the lossless compression performance achieved by the LEC algorithm and recorded in Table 7 were the result of our simulations of the LEC algorithm following the descriptions in the original papers as closely as possible. It can therefore be seen from Table 7 that, our proposed ALDC algorithm outperforms all the other recently proposed lossless compression schemes for WSNs. In addition, a good look at Figure 10 to Figure 16 shows that the compression performance of ALDC for block size of 1 (i.e., when $n = 1$) is quite high (just trailing the performances of LEC) and better than that achieved by the S-LZW algorithm for all the seven data sets. Similarly, with block size as small as say 4 (i.e., $n = 4$), the lossless compression performance achieved by our proposed ALDC algorithm using the decision regions approach is better than that achieved by all the other previously proposed lossless compression algorithms for all the seven data sets.

Sensor nodes transmit data in packets and many systems recommend packet size of not more than 90 bytes. For example, the TinyOS operating systems have sets the default packet payload to 29 bytes. We therefore took advantage of this inherent sensor node transmission mode by collecting source samples in a buffer. We encode the samples in the buffer together. Using the right buffer size (and/or block size), encoding can be done in real-time. Thus, our proposed ALDC algorithm has significant advantages over other lossless compression schemes. While other schemes can only be applied in delay-tolerant applications (e.g., S-LZW) or real-time (delay-intolerant) applications (e.g., LEC), our proposed ALDC scheme can be applicable in both scenarios. Our proposed scheme achieved compression performance up to 74.02% for the real-world datasets.

In terms of algorithm complexity, our proposed ALDC algorithm is simple. When compared to the LEC algorithm, our proposed algorithm requires only slightly more memory. When compared to the S-LZW, our proposed algorithm requires much less memory.

5. Conclusion

In this paper, we have presented a lightweight adaptive lossless data compression algorithm for wireless sensor networks. Our proposed ALDC Scheme performs compression losslessly using two code options. Our proposed ALDC algorithm is efficient and simple, and is particularly suitable for resource-constrained wireless sensor nodes. Our proposed ALDC compression scheme allows compression to dynamically adjust to a changing source. Our proposed algorithm reduce the data amount for transmission which contributes to the energy saving. Additionally, our proposed algorithm can be used in monitoring systems that have different types of data and still provide satisfactory compression ratios. Furthermore, our proposed ALDC algorithm took into account the different real-time requirements on data compression. Thus, our algorithm is suitable for both real-time and delay-tolerant transmission. Our proposed scheme achieved compression performance up to 74.02% using real-world data sets. We also report and analyze using real-world data sets the performance comparisons between our proposed ALDC and other recently proposed lossless compression schemes for WSNs like LEC and S-LZW. We showed that our proposed ALDC algorithm outperforms all the other recently proposed lossless compression schemes. In future, we intend to carry out a formal mathematical modeling and analyses of the Decision Regions Approach of our proposed ALDC algorithm.

References

[1] G. Anastasi, M. Conti, M. Di Francesco, and A. Passarella, "Energy conservation in wireless sensor networks: a survey," *Ad Hoc Networks*, vol. 7, no. 3, pp. 537–568, 2009.

[2] N. Kimura and S. Latifi, "A survey on data compression in wireless sensor networks," in *Proceedings of the International Conference on Information Technology: Coding and Computing (ITCC '05)*, vol. 2, pp. 8–13, April 2005.

[3] F. Marcelloni and M. Vecchio, "An efficient lossless compression algorithm for tiny nodes of monitoring wireless sensor networks," *Computer Journal*, vol. 52, no. 8, pp. 969–987, 2009.

[4] C. Tharini and P. Vanaja Ranjan, "Design of modified adaptive Huffman data compression algorithm for wireless sensor network," *Journal of Computer Science*, vol. 5, no. 6, pp. 466–470, 2009.

[5] J. Yick, B. Mukherjee, and D. Ghosal, "Wireless sensor network survey," *Computer Networks*, vol. 52, no. 12, pp. 2292–2330, 2008.

[6] I. F. Akyildiz and M. C. Vuran, *Wireless Sensor Networks*, John Wiley & Sons, Chichester, UK, 2010.

[7] C. Tharini and P. V. Ranjan, "An efficient data gathering scheme for Wireless Sensor Networks," *European Journal of Scientific Research*, vol. 43, no. 1, pp. 148–155, 2010.

[8] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci,

"Wireless sensor networks: a survey," *Computer Networks*, vol. 38, no. 4, pp. 393–422, 2002.

[9] K. Dolfus and T. Braun, "An evaluation of compression schemes for wireless networks," in *Proceedings of the International Congress on Ultra Modern Telecommunications and Control Systems and Workshops (ICUMT '10)*, pp. 1183–1188, October 2010.

[10] A. van der Byl, R. Neilson, and R. H. Wilkinson, "An evaluation of compression techniques for Wireless Sensor Networks," in *Proceedings of the IEEE Africon*, pp. 1–6, September 2009.

[11] K. C. Barr and K. Asanović, "Energy-aware lossless data compression," *ACM Transactions on Computer Systems*, vol. 24, no. 3, pp. 250–291, 2006.

[12] F. Marcelloni and M. Vecchio, "A simple algorithm for data compression in wireless sensor networks," *IEEE Communications Letters*, vol. 12, no. 6, pp. 411–413, 2008.

[13] T. Srisooksai, K. Keamarungsi, P. Lamsrichan, and K. Araki, "Practical data compression in wireless sensor networks: a survey," *Journal of Network and Computer Applications*, vol. 35, no. 1, pp. 37–59, 2012.

[14] T. Schoellhammer, E. Osterweil, B. Greenstein, M. Wimbrow, and D. Estrin, "Lightweight temporal compression of microclimate datasets," in *Proceedings of the 29th Annual IEEE International Conference on Local Computer Networks (LCN '04)*, pp. 516–524, November 2004.

[15] C. M. Sadler and M. Martonosi, "Data compression algorithms for energy-constrained devices in delay tolerant networks," in *Proceedings of the 4th International Conference on Embedded Networked Sensor Systems (SenSys '06)*, pp. 265–278, November 2006.

[16] S. S. Pradhan, J. Kusuma, and K. Ramchandran, "Distributed compression in a dense microsensor network," *IEEE Signal Processing Magazine*, vol. 19, no. 2, pp. 51–60, 2002.

[17] J. Chou, D. Petrovic, and K. Ramchandran, "A distributed and adaptive signal processing approach to reducing energy consumption in sensor networks," in *Proceedings of the 22nd Annual Joint Conference on the IEEE Computer and Communications Societies*, pp. 1054–1062, April 2003.

[18] A. Ciancio and A. Ortega, "A distributed wavelet compression algorithm for wireless multihop sensor networks using lifting," in *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP '05)*, vol. 4, pp. IV825–IV828, March 2005.

[19] A. Ciancio, S. Pattem, A. Ortega, and B. Krishnamachari, "Energy-efficient data representation and routing for wireless sensor networks based on a distributed wavelet compression algorithm," in *Proceedings of the 5th International Conference on Information Processing in Sensor Networks (IPSN '06)*, pp. 309–316, April 2006.

[20] J. J. Xiao, S. Cui, Z. Q. Luo, and A. J. Goldsmith, "Power scheduling of universal decentralized estimation in sensor networks," *IEEE Transactions on Signal Processing*, vol. 54, no. 2, pp. 413–422, 2006.

[21] J. B. Predd, S. R. Kulkarni, and H. V. Poor, "Distributed learning in wireless sensor networks," *IEEE Signal Processing Magazine*, vol. 23, no. 4, pp. 56–69, 2006.

[22] D. L. Donoho, "Compressed sensing," *IEEE Transactions on Information Theory*, vol. 52, no. 4, pp. 1289–1306, 2006.

[23] E. P. Capo-Chichi, H. Guyennet, and J. M. Friedt, "K-RLE: a new data compression algorithm for wireless sensor network," in *Proceedings of the 3rd International Conference on Sensor Technologies and Applications, (SENSORCOMM '09)*, pp. 502–507, June 2009.

[24] F. Marcelloni and M. Vecchio, "Enabling energy-efficient and lossy-aware data compression in wireless sensor networks by multi-objective evolutionary optimization," *Information Sciences*, vol. 180, no. 10, pp. 1924–1941, 2010.

[25] A. K. Maurya, D. Singh, and A. K. Sarje, "Median predictor based data compression algorithm for Wireless Sensor Network," *International Journal of Smart Sensors and Ad Hoc Networks*, vol. 1, no. 1, pp. 62–65, 2011.

[26] Y. Liang and W. Peng, "Minimizing energy consumptions in Wireless Sensor Networks via two-modal transmission," *Computer Communication Review*, vol. 40, no. 1, pp. 13–18, 2010.

[27] T. A. Welch, "Technique for high-performance data compression," *Computer*, vol. 17, no. 6, pp. 8–19, 1984.

[28] R. F. Rice, P. S. Yeh, and W. Miller, "Algorithms for a very high speed universal noiseless coding module," *JPL Publication Laboratory*, vol. 91, no. 1, pp. 1–30, 1991.

[29] "SensorScope deployments homepage," 2012, http://sensorscope.epfl.ch/index.php/Main_Page.

[30] "TinyNode homepage," 2012, http://www.tinynode.com/.

[31] "Sensirion homepage," 2012, http://www.sensirion.com/.

[32] "Seismic dataset," 2012, http://www-math.bgsu.edu/?zirbel/.