A Simple Data Compression Algorithm for Wireless Sensor Networks

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

[1] Department of Electrical and Electronics Engineering,
The University of Nottingham Malaysia Campus,
Jalan Broga, 43500 Semenyih, Selangor Darul Ehsan, Malaysia
[2] School of Engineering,
Edith Cowan University,
Joondalup, WA 6027, Australia
[3] School of Computer Technology,
Sunway University,
5 Jalan Universiti, Bandar Sunway,
46150 Petaling Jaya, Selangor, Malaysia
{keyx1jgk,Sanandan.Shanmugam,Lim.Wee-Gin}@nottingham.edu.my,
li-minn.ang@ecu.edu.au,
jasmines@sunway.edu.my

Abstract. The energy consumption of each wireless sensor node is one of critical issues that require careful management in order to maximize the lifetime of the sensor network since the node is battery powered. The main energy consumer in each node is the communication module that requires energy to transmit and receive data over the air. Data compression is one of possible techniques that can reduce the amount of data exchanged between wireless sensor nodes. In this paper, we proposed a simple lossless data compression algorithm that uses multiple Huffman coding tables to compress WSNs data adaptively. We demonstrate the merits of our proposed algorithm in comparison with recently proposed LEC algorithm using various real-world sensor datasets.

Keywords: Wireless Sensor Networks, Energy Efficiency, Data Compression, Signal Processing, Adaptive Entropy Encoder, Huffman Coding.

1 Introduction

Wireless sensor networks (WSNs) are very large scale deployments of tiny smart wireless sensor devices working together to monitor a region and to collect data about the environment. Sensor nodes are generally self-organized and they communicate with each other wirelessly to perform a common task. The nodes are deployed in large quantities (from tens to thousands) and scattered randomly in an ad-hoc manner in the sensor field (a large geographic area). Through advanced mesh networking protocols,

* Corresponding author.

V. Snasel et al. (Eds.): SOCO Models in Industrial & Environmental Appl., AISC 188, pp. 327–336.
springerlink.com

these sensor nodes form a wide area of connectivity that extends the reach of cyberspace out into the physical world. Data collected by each sensor node is transferred wirelessly to the sink either directly or through multi-hop communication.

Wireless sensor nodes have limited power source since they are powered by small batteries. 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 depends strongly 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. For this reason, energy-efficient operation should be the most important factor to be considered in the design of WSNs. Thus, several approaches are followed in the literature to address such power limitations. Some of these approaches include adaptive sampling [1], energy-efficient MAC protocols [2], energy-aware routing [3] and in-network processing (aggregation and compression) [4]. 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 [5–7]. One significant approach to conserve energy and maximize network lifetime in WSN is through the use of efficient data compression schemes [8]. Data compression schemes reduce data size before transmitting in the wireless medium which translates to reduce total power consumption. This savings due to compression directly translate into lifetime extension for the network nodes. Both the local single node that compresses the data as well as the intermediate routing nodes benefits from handling less data [9].

Two data compression approaches have been followed in the literature: a distributed data compression approach [10], [11] and a local data compression approach [5], [9], [12–15]. In this paper, our focus is on lossless and reliable data gathering in WSN using a local data compression scheme which has been shown to significantly improve WSN energy savings in real-world deployments [9]. After a careful study of local lossless data compression algorithms (such as Sensor LZW (S-LZW) [9], Lossless Entropy Compression (LEC) algorithm [5], Median Predictor based Data Compression (MPDC) [13] and two-modal Generalized Predictive Coding (GPC) [15]) recently proposed in the literature for WSNs, we found that most of the algorithms cannot adapt to changes in the source data statistics. As a result, the compression performance obtained by these algorithms is not optimal. We therefore propose in this paper a simple lossless data compression algorithm for WSN. The proposed algorithm is adaptive. The algorithm adapts to changes in the source data statistics to maximize performance.

To verify the effectiveness of our proposed algorithm, we compare its compression performance with LEC performance. To the best of our knowledge, till date, LEC algorithm is the best lossless data compression algorithm designed specifically for use

in WSNs. However, 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.

The rest of this article is organized as follows. Section 2 presents our proposed data compression algorithm. In section 3, the proposed algorithm is evaluated and compared with the LEC algorithm using real-world WSN data. Finally, we conclude the paper in section 4.

2 The Compression Algorithm

Our proposed data compression algorithm adopts the followings from the LEC algorithm: (a) to increase the compressibility of the sensed data, we adopt a differential compression scheme to reduce the dynamic range of the source symbols. (b) The basic alphabets of residues are divided into groups whose sizes increase exponentially. The groups are then entropy coded and not unary coded as in the original version of exponential-Golomb code. Thus, the dictionaries used in our proposed scheme are called prefix-free-tables. (c) We also adopt the LEC encoding function because of its simplicity and efficiency. Fig. 1 shows the functional block diagram of our proposed simple data compression algorithm. The algorithm is a two-stage process. In the first stage, a simple unit-delay predictor is used to preprocess the sensed data to generate the residues. That is, for every new acquisition m_i, the difference $d_i = x_i - x_{i-1}$ is computed. While x_i is current sensor reading(s), x_{i-1} is the immediate past sensor reading(s). The difference d_i serves as input to the entropy encoders. In the second stage, two types of entropy encoders are used to better capture the underlying temporal correlation in the sensed data. The entropy encoders are 1-Table Static Entropy Encoder and 2-Table Adaptive Entropy Encoder.

1-Table Static Entropy Encoder

The 1-Table Static Entropy Encoder is essentially the entropy encoder in LEC with its coding table optimized for WSNs sensed data. The encoder performs compression losslessly by encoding differences d_i more compactly based on their statistical characteristics in accordance with the pseudo-code described in Fig. 2. Each d_i is represented as a bit sequence c_i composed of two parts h_i and l_i (that is c_i=h_i*l_i).

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

where

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

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

Equation (3) returns the index position of each d_i within its group. $(\text{Index})|_{\text{bi}}$ 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 to the bit stream which forms the compressed version of the sequence of measures m_i.

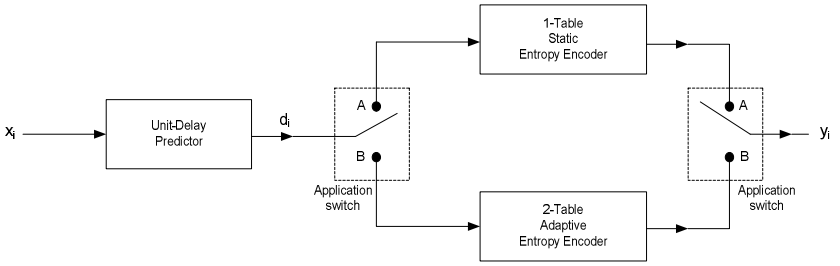

Fig. 1. The functional block scheme of our proposed data compression algorithm

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

// compute di category
IF di = 0 THEN
    SET bi TO 0
ELSE
    // for all bi ≠ 0 then
    SET bi TO ⌈log2(|di|)⌉
ENDIF
// extract hi the variable length Huffman code from TABLE
SET hi TO TABLE[bi]
// build ci
IF bi = 0 THEN
    // li is not needed
    SET ci TO hi
ELSE
    // build li
    SET li TO (Index)|bi
    // build ci
    SET ci TO hi * li
ENDIF
RETURN ci
```

Fig. 2. The pseudo-code of the encode algorithm

2-Table Adaptive Entropy Encoder

High compression ratio performance yields high energy saving since fewer numbers of bits will be transmitted by the communication module of the sensor node thereby saving lots of energy. Adaptive encoding can enable us to achieve maximal compression ratio and by extension maximal energy saving. Hence, to enjoy the benefits of

adaptive coding without incurring higher energy cost, we resorted to the use of multiple static Huffman coding tables. Thus, we propose to implement a 2-Table adaptive entropy encoder that compresses blocks of sampled data at a time using two static Huffman coding tables adaptively. Each static Huffman coding table is designed to give nearly optimal compression performance for a particular geometrically distributed source. By using two Huffman coding tables adaptively and sending the table identifier, the 2-Table adaptive entropy encoder can adapt to many source data with different statistics. The proposed 2-Table adaptive entropy encoder operates in one pass and can be applied to multiple data types. Thus, our proposed algorithm can be used in continuous monitoring systems with varied latency requirements by changing the block size n to suit each application. The pseudo-code of our proposed simple data compression algorithm is given in Fig. 3.

3 Simulations and Analysis

The performance of our proposed data compression algorithm is computed by using the compression ratio defined as:

$$CR = 100 \times \left(1 - \frac{compressed\_size}{original\_size} \right)\% \tag{4}$$

The compressed_size is the number of bits obtained after compression and the original_size is the uncompressed data size. Each uncompressed sample data is represented by 16-bit unsigned integers. Publicly accessible real-world environmental monitoring datasets are used in our simulations. We used temperature and relative humidity measurements from one SensorScope [16] deployments: LUCE Deployment with Node ID of 84 for the time interval of 23 November 2006 to 17 December 2006. We also used the six set of soil temperature measurements from [17], collected from 01 January 2006 to 02 October 2006. For simplicity, the flags of missing data in the soil temperature measurements which are quite rare were replaced by the value of the preceding sample in that soil temperature dataset. To simulate real-world sensor communications with fidelity, the temperature and relative humidity measurements are converted to sensor readings using the inverted versions of the conversion functions in [18] with the assumption of the A/D conversion precision being 14 bits and 12 bits for temperature and relative humidity datasets respectively. In addition, we also used a seismic dataset 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) [19].

```
compress(xi, xi_1, n, y)
// xi is the current sensor reading(s)
// xi_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
// encode() is the encode function

// compute the residue di
SET di TO xi – xi_1
// encode the residue di
IF n = 1 THEN
   // encode residue di using the 1-table static entropy encoder
   CALL encode() with di and Table1 RETURNING ci
   // append ci to y
   SET y TO << y, ci >>
ELSE
   // encode block of n di using the 2-table adaptive entropy encoder
   // encode the block of n di using the first table of the adaptive entropy encoder
   CALL encode() with block of n di and Table2 RETURNING ci
   SET ciA To ci
   // compute the size of the encoded bitstream ciA
   SET size_A TO length(ciA)
   // encode the block of n di using the second table of the adaptive entropy encoder
   CALL encode() with block of n di and Table3 RETURNING ci
   SET ciB To ci
   // compute the size of the encoded bitstream ciB
   SET size_B TO length(ciB)
   // compare size_A and size_B and select the encoded bitstream with the best compressed size
   IF size_A <= size_B THEN
      // generate the table identifier of Table2
      SET ID TO '0'
      // append encoded bitstream ciA to ID
      SET code TO << ID, ciA >>
   ELSE
      // generate the table identifier of Table3
      SET ID TO '1'
      // append encoded bitstream ciB to ID
      SET code TO << ID, ciB >>
   ENDIF
   // append code to y
   SET y TO << y, code >>
ENDIF
RETURN y
```

Fig. 3 The pseudo-code of our proposed simple data compression algorithm

For this simulation, the optimized table in Table 1 is used by the 1-Table static entropy encoder and Tables 2 and 3 were used by the 2-Table adaptive entropy encoder. The Huffman coding table in Table 2 was designed to handle data sets with high and medial correlation while the Huffman coding table in Table 3 was designed to handle data sets with medial and low correlation. The combination of these two coding tables handles effectively the changing correlation in the sensed data as it is being read block by block with each block consisting of *n*-samples. The Huffman table that gives the best compression is then selected. The encoded bitstream generated by that table is then appended to a 1-bit table identifier (ID) and thereafter sent to the sink. The decoder uses the ID to identify the Huffman coding table used in encoding the block of n-residues. Since only two static Huffman coding tables are in use by the 2-Table adaptive entropy encoder, the table ID is either '0' or '1'. The performance comparison between LEC, 1-Table static entropy encoder and 2-Table adaptive entropy encoder is given in Table 4. For block size of 1 (i.e. n=1), our proposed algorithm using the 1-Table static entropy encoder with the optimized coding table givens better performance than the LEC algorithm. For block size greater than 1 (i.e. n>1), our

proposed algorithm using the 2-Table adaptive entropy encoder gives better performance than the LEC algorithm. Thus, the combination of the 1-Table static entropy encoder with the 2-Table adaptive entropy encoder ensures that the performance of our proposed data compression algorithm is better than that of LEC for all value of n.

Table 1. Huffman Coding **Table1**

b_i	h_i	d_i
0	100	0
1	110	−1,+1
2	00	−3,−2,+2,+3
3	111	−7, . . . ,−4,+4, . . .,+7
4	101	−15, . . . ,−8,+8, . . .,+15
5	010	−31, . . . ,−16,+16, . . .,+31
6	0111	−63, . . . ,−32,+32, . . .,+63
7	01101	−127, . . . ,−64,+64, . . .,+127
8	011001	−255, . . . ,−128,+128, . . .,+255
9	0110001	−511, . . . ,−256,+256, . . .,+511
10	01100001	−1023, . . . ,−512,+512, . . .,+1023
11	011000001	−2047, . . . ,−1024,+1024, . . .,+2047
12	01100000000	−4095, . . . ,−2048,+2048, . . .,+4095
13	01100000001	−8191, . . . ,−4096,+4096, . . .,+8191
14	01100000010	−16383, . . . ,−8192,+8192, . . .,+16383

Table 2. Huffman Coding **Table2**

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

Fig. 4 shows the compression ratios achieved by our proposed simple data compression algorithm for different values of the block size n for the nine real-world datasets. As evident from Fig. 4, the compression ratio performance achieved by our proposed data compression algorithm for each of the nine datasets increases with respect to the increase in the block size. Also, for values of n as small as 3 (i.e. n=3), the compression ratio performance of our proposed simple data compression algorithm is good (better than LEC performance) for all the nine datasets and the performance

improves thereafter as n is increased beyond 3 to 256 as evident from the plots in Fig. 4. The best performance of our proposed compression algorithm is recorded in the last column of Table 4 which clearly shows that our proposed simple data compressed algorithm outperforms the LEC algorithm. In terms of algorithm complexity, our proposed algorithm is simple and lightweight. When compared to the LEC algorithm, our proposed algorithm requires only slightly more memory. Energy is conserved since only fewer numbers of bits are transmitted by the communication module of the sensor node.

Table 3. Huffman Coding **Table3**

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

Table 4. Performance comparison between the 2-Table adaptive entropy encoder with LEC and 1-Table static entropy encoder

DATASET	Compression Ratio (CR) %					
	LEC single Table performance	Our Optimized single Table performance	2-Table Adaptive Entropy Encoder performance			2-Table Adaptive Entropy Encoder Best performance
			n=1	n=2	n=3	
LU84 Temp	70.81	71.69	68.01	70.87	71.77	73.48
LU84 Rh	62.86	63.46	60.50	62.92	63.62	64.75
SEISMIC Data	69.72	71.24	67.00	70.12	71.16	72.98
Ts_0 Temp	52.05	52.25	50.92	53.19	54.07	55.76
Ts_5cm Temp	54.55	54.80	52.38	54.70	55.60	57.05
Ts_10cmTemp	54.96	55.21	52.50	55.09	56.00	57.68
Ts_20cm Temp	55.10	55.35	52.51	55.24	56.17	57.97
Ts_50cm Temp	55.04	55.32	52.44	55.22	56.16	57.97
Ts_1m Temp	54.97	55.24	52.40	55.16	56.09	57.88

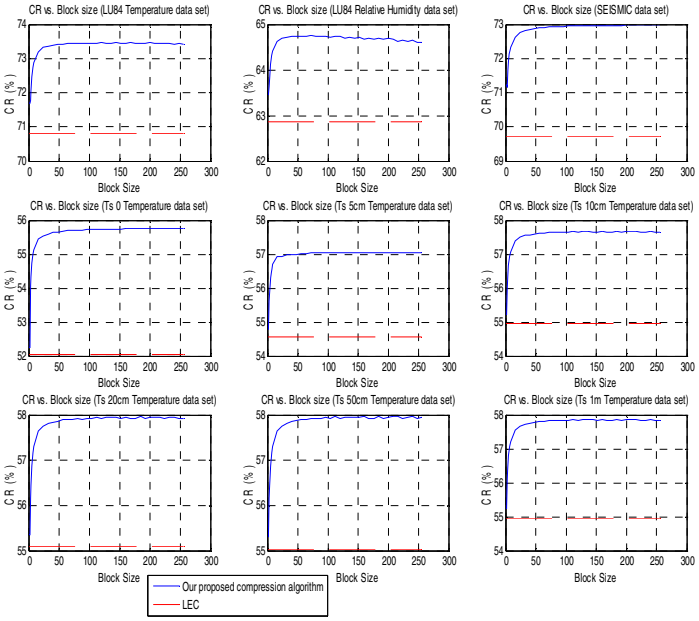

Fig. 4 Compression ratios vs. block size achieved by our proposed data compression algorithm for the nine real-world datasets

4 Conclusion

In this paper, we have introduced a simple lossless adaptive compression algorithm for WSNs. The proposed algorithm is simple and efficient, and is particularly suitable for resource-constrained wireless sensor nodes. The algorithm adapts to changing correlation in the sensed data to effectively compress data using two Huffman coding tables. Our proposed algorithm reduce the data amount for transmission which contributes to the energy saving. We have obtained compression ratios of 73.48%, 64.75% and 72.98% for temperature, relative humidity and seismic datasets respectively. The evaluation of our proposed algorithm with LEC using real-world datasets shows that our proposed algorithm's compression performance is better. Our algorithm can be used for both real-time and delay-tolerant transmission.

References

[1] Bandyopadhyay, S., Tian, Q., Coyle, E.J.: Spatio-Temporal Sampling Rates and Energy Efficiency in Wireless Sensor Networks. IEEE/ACM Transaction on Networking 13, 1339–1352 (2005)

[2] Ye, W., Heidemann, J., Estrin, D.: Medium Access Control With Coordinated Adaptive Sleeping for Wireless Sensor Networks. IEEE/ACM Transactions on Networking 12(3), 493–506 (2004)

[3] Heinzelman, W.R., Chandrakasan, A., Balakrishnan, H.: Energy-efficient communication protocol for wireless microsensor networks. In: Proceedings of the 33rd Annual Hawaii International Conference on System Sciences (2000)
[4] Fasolo, E., Rossi, M., Zorzi, M., Gradenigo, B.: In-network Aggregation Techniques for Wireless Sensor Networks: A Survey. IEEE Wireless Communications 14, 70–87 (2007)
[5] Marcelloni, F., Vecchio, M.: An Efficient Lossless Compression Algorithm for Tiny Nodes of Monitoring Wireless Sensor Networks. The Computer Journal 52(8), 969–987 (2009)
[6] Anastasi, G., Conti, M., Di Francesco, M., Passarella, A.: Energy conservation in wireless sensor networks: A survey. Ad Hoc Networks 7(3), 537–568 (2009)
[7] Barr, K.C., Asanović, K.: Energy-aware lossless data compression. ACM Transactions on Computer Systems 24(3), 250–291 (2006)
[8] Yick, J., Mukherjee, B., Ghosal, D.: Wireless sensor network survey. Computer Networks 52(12), 2292–2330 (2008)
[9] Sadler, C.M., Martonosi, M.: Data compression algorithms for energy-constrained devices in delay tolerant networks. In: Proceedings of the 4th international conference on Embedded networked sensor systems - SenSys 2006, p. 265 (2006)
[10] Ciancio, A., Pattem, S., Ortega, A., Krishnamachari, B.: Energy-Efficient Data Representation and Routing for Wireless Sensor Networks Based on a Distributed Wavelet Compression Algorithm. In: Proceedings of the Fifth international Conference on Information Processing in Sensor Networks, pp. 309–316 (2006)
[11] Gastpar, M., Dragotti, P.L., Vetterli, M.: The Distributed Karhunen-Loeve Transform. IEEE Transactions on Information Theory 52(12), 5177–5196 (2006)
[12] Schoellhammer, T., Greenstein, B., Osterweil, E., Wimbrow, M., Estrin, D.: Lightweight temporal compression of microclimate datasets [wireless sensor networks. In: 29th Annual IEEE International Conference on Local Computer Networks, pp. 516–524 (2004)
[13] Maurya, A.K., Singh, D., Sarje, A.K.: Median Predictor based Data Compression Algorithm for Wireless Sensor Network. International Journal of Smart Sensors and Ad Hoc Networks (IJSSAN) 1(1), 62–65 (2011)
[14] Kolo, J.G., Ang, L.-M., Seng, K.P., Prabaharan, S.: Performance Comparison of Data Compression Algorithms for Environmental Monitoring Wireless Sensor Networks. International Journal of Computer Applications in Technology, IJCAT (article in press, 2012)
[15] Liang, Y.: Efficient Temporal Compression in Wireless Sensor Networks. In: 36th Annual IEEE Conference on Local Computer Networks (LCN 2011), pp. 466–474 (2011)
[16] SensorScope deployments homepage (2012), `http://sensorscope.epfl.ch/index.php/Main_Page` (accessed: January 2012)
[17] Davis, K.J.: No Title (2006), `http://cheas.psu.edu/data/flux/wcreek/wcreek2006met.txt` (accessed: January 6, 2012)
[18] Sensirion homepage, (2012), `http://www.sensirion.com` (accessed: January 6, 2012)
[19] Seismic dataset (2012), `http://www-math.bgsu.edu/?zirbel/` (accessed: January 6, 2012)