COMPRESSION AND PROGRESSIVE RETRIEVAL OF MULTI-DIMENSIONAL SENSOR DATA

P. Lorkowski [a, *], T. Brinkhoff [a]

[a] Institute for Applied Photogrammetry and Geoinformatics (IAPG),
Jade University Wilhelmshaven/Oldenburg/Elsfleth, Ofener Str. 16/19, D-26121 Oldenburg, Germany -
(peter.lorkowski, thomas.brinkhoff)@jade-hs.de

Commission II, WG II/1

KEY WORDS: Continuous Phenomena, Discrete Observations, Binary Space Partitioning

ABSTRACT:

Since the emergence of sensor data streams, increasing amounts of observations have to be transmitted, stored and retrieved. Performing these tasks at the granularity of single points would mean an inappropriate waste of resources. Thus, we propose a concept that performs a partitioning of observations by spatial, temporal or other criteria (or a combination of them) into data segments. We exploit the resulting proximity (according to the partitioning dimension(s)) within each data segment for compression and efficient data retrieval. While in principle allowing lossless compression, it can also be used for progressive transmission with increasing accuracy wherever incremental data transfer is reasonable. In a first feasibility study, we apply the proposed method to a dataset of ARGO drifting buoys covering large spatio-temporal regions of the world´s oceans and compare the achieved compression ratio to other formats.

1. INTRODUCTION

Data compression is one key aspect of managing sensor data streams, since technological progress about transfer rate, processing power and memory size tends to be outperformed by the ever-growing amount of available observations. The increased mobility of sensors due to miniaturization and improved energy efficiency extends their capabilities. On the other hand, it requires more advanced techniques of data processing and analysis to exploit these new opportunities. For achieving high efficiency, compression methods should take into account the specific structure of the data they are applied to.

Sensor observations typically describe continuous or quantitative variables in multiple dimensions like latitude and longitude, time, temperature, pressure, voltage, etc. Where these data, at least locally, tend to be stationary in space and time, there is high potential for compression: the actual values within a period and/or spatial range usually cover only a small range of values compared to the domain represented by a standard data type like floating-point numbers. Thus, we propose to perform a partitioning of observations by spatial, temporal or other criteria (or a combination of them) into data segments. For data retrieval from spatial or spatio-temporal databases, the creation of such data segments is also reasonable.

One central feature of our concept is that it supports progressive data loading for applications that do not (immediately) need the full accuracy of the queried data. This is especially useful for environments with limited transmission rate, image resolution and processing power like mobile computing. We propose to use recursive binary subdivision of the multidimensional value space for this purpose. For a given level of progression, an identical accuracy (relative to the range of values) can be achieved for each dimension. When using a database as a sink, it is reasonable to store those data segments as BLOBs indexed by the dimension(s) used for partitioning. Queries defined by (spatio-temporal) bounding boxes will then be processed in two stages: First, the data segments affected by the query are identified. In the second step, the data segments are progressively decoded and transmitted until the required accuracy (e.g. for scientific analysis, web mapping or mobile computing) is reached.

The remainder of this article is structured as follows: In the next section, we provide an overview of related work dealing with sensor data compression. We introduce the methodology used to compress multi-dimensional sensor data in Section 3. The methodology will also be concretized for the different supported data types here before in Section 4 we propose some extensions that are not yet implemented but which we think might be useful. Section 5 provides the results from our experimental study with the data of ARGO drifting buoys monitoring sea the surface temperature and salinity of the world´s oceans. Finally, we draw some conclusions in Section 6.

2. RELATED WORK

Managing sensor data streams has to cope with various limitations of resources like energy, transmission rate, computational power, volatile and non-volatile memory. In many cases, compression of observational data can help to overcome those limitations. Thus, it is widely discussed in literature.

Medeiros et al. (2014) suggest a Huffman encoding applied to differences of consecutive measurements and thus achieve high compression ratios. This method works very efficient with time

* Corresponding author

series of single sensors for one dimension with small changes between consecutive observations.

A more adaptive approach of Huffman encoding is introduced by Kolo et al. (2012), where data sequences are partitioned into blocks which are compressed by individual schemes for optimized efficiency.

Sathe et al. (2013) introduce various compression methods, mainly known from the signal processing literature. Those are restricted to one measurement variable of one sensor.

Dang et al. (2013) propose a virtual indexing to cluster measurements that are similar in value but not necessarily spatio-temporally proximate. After this rearrangement, the data are compressed using discrete cosine transformations and discrete wavelet transformations.

The compression of multidimensional signals is covered by (Duarte & Baraniuk, 2012) and (Leinonen et al., 2014). Both works apply the Kronecker compressive sensing approach exploiting sparse approximation of signals with matrix algebra and is of high computational complexity.

The works listed above make use of the often strong correlation of consecutive sensor measurements for compression.

The contribution of our compression method in this context is to address the following requirements simultaneously:

- The compressed units of data are organized as spatio-temporally confined segments suited for systematic archival in spatial/spatio-temporal databases.
- Diverse data types such as *Double*, *Integer*, *DateTime*, *Boolean* can be compressed losslessly.
- Compression/decompression of multiple data dimensions is performed simultaneously.
- Within one data segment, observations are compressed independently (no consecutive observations of single sensors tracked by their IDs are considered) and thus can handle data from mobile sensors that are arbitrarily distributed in space and time.
- Data can be decoded progressively, e.g. for preview maps or applications with limited accuracy demands.
- Computational costs for coding/decoding are low.

3. METHODOLOGY

3.1 Principle and General Design

The principle applied for our compression method is derived from the Binary Space Partitioning tree (BSP tree), see (Samet, 2006). Unlike its common utilization for indexing, we use it as compression method applied to each single observation in a dataset. It does not presume high correlation of consecutive observations (time series) like e.g. Huffman encoding does (Kolo et al., 2012, Medeiros et al., 2014). Thus, our algorithm does not need to keep track of individual sensors within a set of observations, but encodes each observation individually within the given value domains.

The general idea behind the design is to encode observations describing a continuous phenomenon within a (spatio-temporal) region. We focus on the representation of the continuous field as a whole, not on the time series of individual sensors. This in mind, it appears reasonable to filter out observations that do not significantly contribute to the description of the field before long-term archival of the data. When embedded into a monitoring system, our approach will perform best after some deliberate depletion based on spatio-temporal statistics (Lorkowski & Brinkhoff, 2015).

Progressive decompression can support different requirement profiles and is thus another important design feature of our approach. For some applications, it might be reasonable to prioritise response time behaviour (at least for first coarse results) against full accuracy after performing one step. The specific structure of our binary format supports this claim.

3.2 Binary Interval Subdivision

For each n-dimensional set of observational data, an n-dimensional minimum-bounding box over the values actually can be calculated. In the following, the minimum and the maximum value of a dimension are denoted by *min* and *max*, respectively. The interval [*min*,*max*] will be called *value domain*. The value domain is a subset of the domain covered by the corresponding data type.

Assuming the region of interest to be spatially and/or temporally confined and the phenomena observed to be typically stationary like temperature, there is a good chance for the value domain to be relatively small. Thus, we can achieve a high resolution requiring relatively few bits of data by using the multi-dimensional recursive binary region subdivision. The principle for one dimension is depicted in Fig. 1, where an interval is recursively partitioned by the binary sequence 0-1-1. The circle with double arrow represents the position with its maximum deviation defined by that particular sequence of subdivision steps (in the following also called *levels*) within the interval.

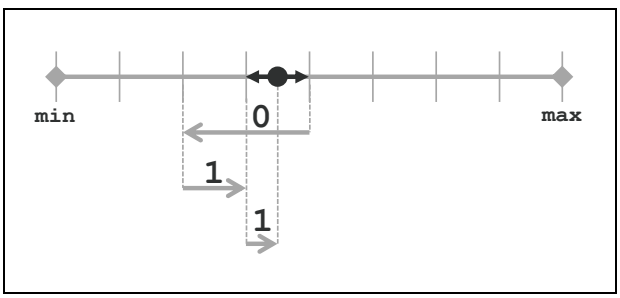

Figure 1. Binary interval subdivision

As can easily be concluded from Fig. 1, the number of necessary bits depends on the required absolute accuracy as well as on the size of the particular value domain within the dataset to be compressed.

The considerations above show general a one-dimensional spatial perspective on the problem. Since we work with sensor data streams, we have to apply this principle to specific data types common in this context.

3.3 Supported Data Types

In a sensor web environment, the collected data can in principle be of ordinal scale (e.g., sensor number), nominal scale (type of

material), discrete scale (number of neighbouring sensors) or continuous scale (time, temperature) (McKillup & Dyar, 2010).

In the domain of data management and programming, this information is typically represented by the data types *Integer*, *Float* or *Double*, *Boolean* and *DateTime*. Within a dataset or observation epoch, the actual data range is usually only a small fraction of the range covered by the appropriate data type.

Since the data types mentioned above have different characteristics, those will have to be considered when applying the multi-dimensional progressive compression.

Double/Float:
The compression is most straightforward for this data type. The binary tree depth *n* can be determined by:

$$n = log_2(d/a) \quad (1)$$

where d = value domain (max – min)
a = accuracy or maximum deviation

Within a multi-dimensional setting, the *relative accuracy* of each dimension is equal for equal *n* while the *absolute accuracy* also depends on the size of its value domain.

Thus, when performing the compression synchronously for all dimensions with each step or level, as we suggest here, we achieve equal relative accuracy for each dimension. This does not apply when one dimension has already reached its maximum bit depth while others still have not (see Tab. 1) or when particular dimensions have more than one bit per level to achieve faster convergence (see Section 3.5).

In the case of a *Float/Double* data type, the interval depicted in Fig. 1 directly represents the minimum and maximum of the value domain, and the double arrow represents the accuracy or maximum deviation reached at the particular level (here: 0-1-1).

Integer:
Although at first glance the data type *Integer* is the less complex, it is in fact somewhat more difficult to handle in respect of progressive compression. First, we have to carefully avoid the fencepost error when compressing/decompressing to the last level. So if an interval shall represent 3 integer segments as depicted in Fig. 2, we need to enlarge it by one before calculating the extent (max - min) to achieve their correct representation on the scale.

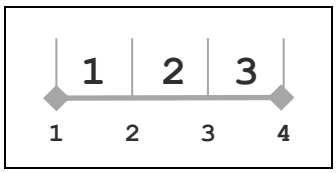

Figure 2. Fencepost error problem

If *Integer* numbers are used for nominal scales (e.g. for IDs), coarse indications within the value domain are maybe rather useless. For that reason it might be necessary to evaluate to the complete bit depth. If a nominal value domain requires the highest number of bits to be represented (as column 6 in Tab. 1), all data will have to be transmitted completely before the *Integer* number of this dimension is resolved. For more flexibility, we allow individual bit lengths per step or level for each dimension (see Section 3.5).

Boolean:
Boolean values can be seen as a special case of Integers with a range of 2. The special thing about *Boolean* values is that we always need only one step or level to express the one bit of information (last column in Tab. 1).

DateTime:
Unlike the *Integer* type, the *DateTime*-type appears much more complex at first glance than it is in handling. This is the case because it can be interpreted (and also is usually represented internally) as *ticks* (e.g. 100 nanoseconds) elapsed since some reference point in time (e.g. 01.01.0001, 00:00:00 h). This internal value (usually a 64-bit integer) is provided by most libraries and can be used to handle the *DateTime* data type as normal *Integer* or *Double* for compression. Usually, time spans within a dataset of observations are tiny compared to the one covered by *DateTime*, and the necessary temporal resolution is also by far lower than that of this data type. Thus, we can expect high compression rates for this data type.

3.4 Parallel compression of all Dimensions

One central feature of our compression format is the progressive retrieval of sets of observations with increasing accuracy with each step or level. The general format is shown in Tab. 1 that displays the compression format for 9 dimensions of one observation.

```
101011001
10100000
01001100
11000010
01001000
01100000
011010 0
100001 0
010001 0
111001 0
0010 1
00 0 1
01 0 0
10 0 1
0  1 1
     0
     1
     1
```

Table 1. Binary format for progressive sensor data storage

Each column represents one dimension and each row stands for one level of progressive coding/decoding. The bitstream of a particular dimension terminates at the level where its preset resolution/accuracy is reached. For the data type *Boolean* (right column) this is already the case after the first step or row.

Unlike the structure displayed in Tab. 1 for visualization, the actual binary format does not contain blank positions, but only the data bits. Therefore, for decompressing it is necessary to consider the exact format stucture to have each bit assigned to the correct dimension.

Due to its general structure, with increasing row numbers this format tends to decrease in data volume per row and finally contributes to the accuracy of the dimensions with highest predefined resolutions only.

3.5 Flexible Bit Length per Row

Given the structure described above can lead to the situation where a particular dimension might not be determined at desired accuracy until the last rows are reached. Most of the data might have been transmitted unnecessarily when for the other dimensions rather low accuracy would have sufficed. This situation might particularly be the case for IDs or nominal scales. It might be indispensable to receive their exact value at an early stage of the stepwise transmission.

As a solution for this problem we suggest that bit lengths per row can be set individually for each dimension. Thus, the value of a dimension can converge much quicker against its actual value with each step. In the extreme case, the exact value can already be provided with the first step of transmission as it is always the case for binary values. This option can be useful when the IDs of observations are needed immediately for visualization or mapping with other data sources.

3.6 Progressive Decompression

As a consequence of the special data structure introduced so far, decompression must permanently keep track of the actual bit configuration in combination with the exact number of bits sent so far. With each new row transmitted, there is an improvement of accuracy (the factor depending on the number of bits per row) for each dimension.

In an environment with bandwidth restrictions, this progressive method provides immediate coarse results, e.g. for visualization. With the last step, the data is transmitted completely lossless, according to the predefined resolution. This is not always necessarily the best choice since the data might not be needed in full accuracy but rather within shorter transmission time. Thus, the transmission can be aborted at any level.

4. EXTENSION

Our compression method as introduced so far fulfils the requirements mentioned in Section 2. There are, however, some ideas not yet implemented but certainly worth considering to be realized in future.

In the buoy data sample introduced in the next section, we find missing measurement values indicated by '999.9999' (see header in Tab. 2). The idea behind this number is to have an optical pattern immediately recognizable for the human eye as exception. Using it as *unset*-indicator within our compression algorithm is rather awkward, since, by being an absolute outlier, it enlarges the value domain (and therefore the necessary bit depth) significantly. A more explicit variant is desirable here, e.g. by indicating validity/invalidity of a value by its first bit. In case of invalidity, the bits to follow for that particular dimension can simply be dropped, but on the other hand, this mechanism would make decoding more complex and would only pay off when having a significant amount of unset values.

Another aspect worth considering is the compression of the header associated with each compressed data segment (see Tab. 3). With its metadata for each value dimension (name, deviation, bit depth, bits per row, min, max) it is crucial for archival and retrieval and a prerequisite for decoding. In a monitoring scenario with very small data segments to be compressed, the relative size of that header can justify its compression if transmission of data is expensive.

Wherever the transmission of the compressed data is potentially error prone and not secured by other protocols, it might become necessary to implement some checksum method within the binary format itself. In this case, the gain in reliability needs to be weighted carefully against the overhead according to implementation, processing and data volume.

5. EXPERIMENTAL STUDY

For our case study with experimental data, we use the drifting buoy data from the ARGO program provided by a Canadian governmental service[1]. The data can be obtained in different formats. The format we chose for our experiments is described in Tab. 2.

```
   Contents:
Col  1 = Platform identifier (ARGOS #)
Col  2 = EXP$ - The originator's experiment number
Col  3 = WMO$ - WMO platform identifier number
Col  4 = Position year/month/day hour:minute (UTC)
Col  5 = Latitude of observation (+ve North)
Col  6 = Longitude of observation
         (+/- 180 deg +ve West of Greenwich)
Col  7 = Observation year/month/day hour:minute (UTC)
Col  8 = SSTP - Sea surface temperature (deg. C)
Col  9 = Drogue on/off - 1 = attached; 0 = not

Note: Missing value indicated by 999.9999
```

Table 2. Header of ARGO drifting buoy data

The sample contains all data types we mentioned in Section 3.3. We find *Integer* types for the IDs in columns 1, 2 and 3. Colums 4 and 7 contain *DateTime* types, columns 5, 6 and 8 represent *Double* numbers while column 9 displays an on/off state as *Binary*.

For our first study we chose a subset of 100, 1000 and 10000 points filtered out by spatial and temporal bounds. Table 3 depicts a corresponding data header generated by our compression program (note the changed names and order compared to Tab. 2). The values for *min* and *max* are derived from the data. Together with the preset value *max_dev* for the maximum deviation the bit depth *bits* is determined using formula (1) in Section 3.3. The value *bpr* indicates the number of bits per row used for each column.

A maximum deviation of 0.5 for integer numbers means that at full bit depth the exact number is provided. For the *DateTime* type this value represents seconds, so the minutes are decoded correctly when set to 30.

fname	max_dev	bits	bpr	min	max
x	0,0005	16	1	10,767	49,671
y	0,0005	15	1	40,07	59,08
val	0,0005	14	1	5,529	18,55
idarg	0,5	16	1	37411	92885
idexp	0,5	12	1	6129	9435
idwmo	0,5	23	1	1300518	6200926
tpos	30	10	1	2010-12-31 21:54	2011-01-01 09:24
tobs	30	10	1	2011-01-01 00:05	2011-01-01 09:57
drg	0	1	1	False	True

Table 3. Header for a dataset of ARGOS drifting buoy observations

[1] http://www.meds-sdmm.dfo-mpo.gc.ca/isdm-gdsi/drib-bder/svp-vcs/index-eng.asp

As can be seen, the value for "idwmo" has the highest bit depth of 23, since the value range is nearly 5 million.

The effect of this is the longest chain of bits for that dimension in the corresponding data file (see Tab. 4).

iii	iii	iii
dddtt	dddtt	dddtt
vaewpod	vaewpod	vaewpod
arxmobr	arxmobr	arxmobr
xylgpossg	xylgpossg	xylgpossg
101011001	010101000	111101000
10100000	01110000	00010000
01001100	00100100	00000100
11000010	00010010	00010010
01001000	01001000	00101000
01100000	01111000	01011001
01101000	01001001	00101000
10000100	01101100	11101101
01000110	11000111	10000111
11100100	00010100	11110100
001001	101011	111011
000001	010111	011111
0100 0	0101 0	1010 0
1010 1	0111 1	0001 1
01 1 1	10 0 1	10 1 1
1 0 0	1 0 1	0 1 0
1	0	1
1	0	0
1	1	0
1	0	0
1	0	0
1	0	0
0	1	1

Table 4. Compressed data for 3 observations of ARGO drifting buoys (column names to be read vertically!)

Three observations are listed, each containing all 9 data columns organized vertically (as are the column names) with increasing accuracy from top to bottom. As can be seen, the binary value of the rightmost field *drg* (indicating drogue on/off) is already complete in the first row whereas the one for *idwmo* is resolved in row 23, as indicated in the header file (Tab. 3).

Since this column represents an ID, it might very likely be necessary to resolve it earlier than in the last data row. Therefore, we increase the number of bits per row to 4. The resulting structure for the same data can be seen in Tab. 5.

ii**i**		ii**i**		ii**i**	
dd**d**	tt	dd**d**	tt	dd**d**	tt
vae**w**	pod	vae**w**	pod	vae**w**	pod
arx**m**	obr	arx**m**	obr	arx**m**	obr
xylgp**o**	ssg	xylgp**o**	ssg	xylgp**o**	ssg
10001**1010**001		01010**1010**000		10101**1010**000	
10000**0001**00		01000**0001**00		11100**0001**00	
01001**1111**00		00010**1111**00		00101**1111**00	
01000**0110**00		00010**0111**00		00100**0110**00	
11001**1111**00		01011**0010**00		01101**0101**10	
11000**110** 00		01011**001** 00		11100**100** 00	
11101	10	01011	10	11101	10
10000	10	11111	10	01100	00
11110	00	11010	00	10100	11
11000	10	10100	11	01100	11
00010	10	10011	11	01100	11
10110	00	01101	01	00100	01
1111		1110		0010	
0010		1101		1110	
0110		0001		1110	
1 10		1 11		1 10	
0		0		1	
1		1		1	
1		1		1	
0		1		1	

Table 5. Compressed data with prolonged bit length per row for column *idwmo* (printed in bold)

In this configuration, the exact values for *idwmo* (printed bold) are already resolved in row 6. In practice, the bit length per row can either be set directly (column *bpr* in the header), determined by maximum number of rows for all bits, or by some arbitrary combination of accuracy and row in the form *in row x accuracy y must be met*. This configuration can be set individually for each dimension to achieve a good balance between stepwise accuracy improvement and total size per data row.

5.1 Results

To create indicators for the performance of our compression method, we have applied it to a dataset of 100, 1,000 and 10,000 observations given in the format described in the section above. We compare four indicators here: The first indicator is the size of the text file as received from the Canadian governmental service provider (denoted by "Text" in the following). The amount of memory necessary when the data is parsed and translated into native machine data types is evaluated as second indicator ("Native"). We assume 32 bits for *Integer*, 64 bits for *Double*, 64 bits for *DateTime* and 8 bits for *Boolean*. Our binary BSP format is the third format listed. (We did not consider the size of the header here.) Finally, we applied ZIP compression of the text file as forth format with 7-Zip with following settings: normal compression level, deflate method, 32 KB dictionary size and a word size of 32.

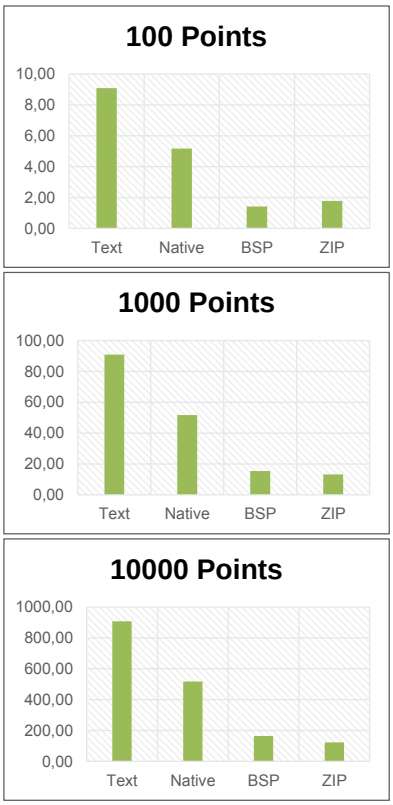

Figure 6. Data volumes (KiB) in different formats for 3 datasets

As can be seen in Fig. 6, our approach outperforms the ZIP compression for the small sample. With growing data size, the efficiency of the ZIP dictionary is increasing, which is not the case for our approach. Nevertheless, taking into account progressive decoding as an important key feature, the slightly worse compression ratio for large datasets appears acceptable.

More valuable experiments with realistic settings for coarse data retrieval will be addressed in future work. The focus here was to introduce the concept and give a rough estimate of its overall efficiency.

6. CONCLUSIONS

The methodology presented here is useful for situations where massive sensor data need to be compressed in a way that allows a progressive retrieval with increasing accuracy per step. It supports the most typical data types found in sensor data like *Float*/*Double*, *Integer*, *Boolean*, and *DateTime*, each one with specific compression methodologies. The compression ratio depends on the value range and necessary accuracy. The number of bits per transmission step can be set in accordance with the transmission priorities, e.g. if certain dimensions are needed with higher convergence of accuracy per step.

The method requires some overhead for communication between data nodes. So the header that determines the mode of transmission needs to be exchanged. In environments where transmission of data is significantly more expensive than processing coding/decoding tasks, this method is likely to pay off. For using the proposed method in a real-time environment, some protocol needs to be created to retain efficiency of transmission: values of defect sensors can be omitted, changed value ranges need to be adjusted and maybe the bits-per-row configuration should be changed due to changed priorities. All this means considerable overhead which should carefully be weighed against achievable savings for data transmission.

When thinking about long-term archival of data streams in databases, there are several points to be considered. Maybe the most important one is how a large dataset is to be segmented into smaller units. Doing this by spatial and/or temporal boundaries is reasonable since this is the most obvious means to refer the sensor data to other aspects like e.g. traffic density. Databases today widely support efficient management of spatial and spatio-temporal data (Brinkhoff, 2013). But the associated indexing techniques were primary developed having retrieval performance and not compression in mind. Thus, it appears reasonable to make use of them at a higher granularity level than the individual observation. So the method proposed here can be applied to appropriate segments of data while using the spatial or spatio-temporal boundaries of that segments for indexing with common database techniques. The compressed segment can be stored as binary large objects (BLOBs) in the database with associated spatial/spatio-temporal index and metadata.

Since the spatio-temporal boundaries can also be seen as statistical properties of the dataset, it is reasonable to ask if additional statistical properties like mean value, standard deviation or skewness should not also be considered for each dataset. This might be of little use for the dimensions space and time, but can be crucial for measured values like temperature or air pollutants. If advanced analysis methods like geostatistics are used, more complex statistical indicators like variogram model parameters should be considered (Lorkowski & Brinkhoff, 2015). All those data should be stored as metadata alongside with each dataset to support efficient retrieval.

One central issue here is the way large datasets are subdivided into smaller subsets on which the compression method is applied to and the corresponding metadata are related to. A good configuration balances retrieval granularity, subset management overhead, indexing costs, transmission data volume, system responsiveness and accuracy in a way that fulfils the requirements of the monitoring system.

REFERENCES

Brinkhoff, T., 2013. *Geodatenbanksysteme in Theorie und Praxis: Einführung in objektrelationale Geodatenbanken unter besonderer Berücksichtigung von Oracle Spatial*. Wichmann, Heidelberg.

Duarte, M. F., Baraniuk, R. G., 2012. Kronecker Compressive Sensing. *IEEE Transactions on Image Processing*, Vol. 21, No. 2, February 2012.

Kolo, J. G., Shanmugam, S. A., Lim, D. W. G., Ang, L.-M., Seng, K. P., 2012. An Adaptive Lossless Data Compression Scheme for Wireless Sensor Networks. *Journal of Sensors*, Volume 2012, Article ID 539638, 20 pages.

Leinonen, M., Codreanu, M., Juntti, M., 2014. Compressed Acquisition and Progressive Reconstruction of Multi-Dimensional correlated Data in Wireless Sensor Networks. *Proceedings IEEE International Conference on Acoustic, Speech and Signal Processing* (ICASSP), Florence, pp. 6449-6453.

Lorkowski, P., Brinkhoff, T., 2015. Environmental Monitoring of Continuous Phenomena by Sensor Data Streams: A System Approach based on Kriging. *Proceedings 29th International Conference on Informatics for Environmental Protection*, Copenhagen, Denmark, Atlantis Press.

McKillup, S., Dyar, M. D., 2010. *Geostatistics Explained: An Introductory Guide for Earth Scientists*. Cambridge University Press, New York.

Medeiros, H. P., Maciel, M. C., Souza, R. D., Pellenz, M. E., 2014. Lightweight Data Compression in Wireless Sensor Networks Using Huffman Coding. *International Journal of Distributed Sensor Networks*, Volume 2014, Article ID 672921, 11 pages.

Samet, H., 2006. *Foundations of Multidimensional and Metric Data Structures*. Morgan Kaufmann, San Francisco.

Sathe, S., Papaioannou, T. G., Jeung, H., Aberer, K., 2013. A Survey of Model-Based Sensor Data Acquisition and Management. In: Aggarwal, C. C. (ed.), *Managing and Mining Sensor Data*, Springer, New York.