MULTI-DIMENSIONAL DATA STREAM COMPRESSION FOR EMBEDDED SYSTEMS

BO LI

A THESIS
IN
THE DEPARTMENT
OF
COMPUTER SCIENCE

PRESENTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF MASTER OF SCIENCE (COMPUTER SCIENCE)
CONCORDIA UNIVERSITY
MONTRÉAL, QUÉBEC, CANADA

AUGUST 2019

Concordia University

School of Graduate Studies

This is to certify that the thesis prepared

By:	**BO LI**
Entitled:	**Multi-dimensional data stream compression for embedded systems**

and submitted in partial fulfillment of the requirements for the degree of

Master of Science (Computer Science)

complies with the regulations of this University and meets the accepted standards with respect to originality and quality.

Signed by the final examining commitee:

________________________________ Chair
Dr. Olga Ormandjieva

________________________________ Examiner
Dr. Emad Shihab

________________________________ Examiner
Dr. Brigitte Jaumard

________________________________ Supervisor
Dr. Tristan Glatard

Approved ________________________________
Chair of Department or Graduate Program Director

__________ 20 ______ ________________________________

Interim Dean

Faculty of Engineering and Computer Science

Abstract

Multi-dimensional data stream compression for embedded systems

BO LI

The rise of embedded systems and wireless technologies led to the emergence of the Internet of Things (IoT). Connected objects in IoT communicate with each other by transferring data streams over the network. For instance, in Wireless Sensor Networks (WSNs), sensor-equipped devices use sensors to capture properties, such as temperature or accelerometer, and send 1D or nD data streams to a host system. Power consumption is a critical problem for connected objects that have to work for a long time without being recharged, as it greatly affects their lifetime and usability. Data summarization is key for energy-constrained connected devices, as transmitting fewer data can reduce energy usage during transmission. Data compression, in particular, can compress the data stream while preserving information to a great extent. Many compression methods have been proposed in previous research. However, most of them are either not applicable to connected objects, due to resource limitation, or only handle one-dimensional streams while data acquired in connected objects are often multi-dimensional. Lightweight Temporal Compression (LTC) is among the lossy stream compression methods that provide the highest compression rate for the lowest CPU and memory consumption. In this thesis, we investigate the extension of LTC to multi-dimensional streams. First, we provide a formulation of the algorithm in an arbitrary vectorial space of dimension n. Then, we implement the algorithm for the infinity and Euclidean norms, in spaces of dimension 2D+t and 3D+t. We evaluate our implementation on 3D acceleration streams of human activities, on Neblina, a module integrating multiple sensors developed by our partner Motsai. Results show that the 3D implementation of LTC can save up to 20% in energy consumption for slow-paced activities, with a memory usage of about 100 B. Finally, we compare our method with polynomial regression compression methods in different dimensions. Our results show that our extension of LTC gives a higher compression ratio than the polynomial regression method, while using less memory and CPU.

Acknowledgments

I would first like to express my deep gratitude to my supervisor, Dr. Tristan Glatard, for his patient guidance, enthusiastic encouragement, and valuable suggestions on this thesis.

I would like to acknowledge the staff of Motsai company for their valuable technical support and assistance of the data collection.

In addition, I am particularly grateful for the help given by Omid Sarbishei and Hosein Nourani in collecting data.

Finally, I wish to thank my parents and my friends for their support and encouragement throughout my study.

Contents

List of Figures

List of Tables

Acronyms

A-ARMA	Adaptive Auto-Regression Moving-Average. 14, 15, 16
Accelerometer-LZSS, A-LZSS	Accelerometer-Lempel-Ziv-Storer-Szymanski. 10
ADC	Analog-to-Digital Converter. 9
AMS Sketch	Alon-Matias-Szegedy Sketch. 8
AR	Auto-Regression. 14, 16
ARMA	Auto-Regression Moving-Average. 14, 15, 16
BLE	Bluetooth Low Energy. 4, 6
Enhanced PLAMLis	Enhanced Piece-wise Linear Approximation with Minimum number of Line Segments. 13, 16
FM-Sketch	Flajolet and Martin Sketch. 8
IoT	Internet of Things. iii, 3, 4, 8
LEC	Lossless Entropy Compression. 9
LTC	Lightweight Temporal Compression. iii, vi, vii, 6, 7, 11, 16, 17, 18, 19, 20, 22, 24, 28, 40, 41
LZ77	Lempel-Ziv-77. 10
LZ78	Lempel-Ziv-78. 10
LZO	Lempel-Ziv-Oberhumer. 10

LZSS Lempel-Ziv-Storer-Szymanski. 10

LZW Lempel-Ziv-Welch. 10

MA Moving-Average. 14

MA-AR Modified Adaptive Auto-Regression. 16

PLAMLis Piece-wise Linear Approximation with Minimum number of Line Segments. 11, 13, 16

RFID Radio Frequency Identification. 3, 4

RMSE Root-Mean-Square Error. 14, 15, 16

Sensor-LZW, S-LZW Sensor-Lempel-Ziv-Welch. 10

S-LEC Sequential Lossless Entropy Compression. 9

WSNs Wireless Sensor Networks. iii, 3, 4

Chapter 1

Introduction

1.1 Context

With the recent technological advances of the Internet of Things (IoT) applications, the number of connected devices will reach 75 billion by 2025[1]. So far, the IoT has been involved in many fields such as medical care, military, sports and industrial manufacturing [7, 23, 12]. Systems embedded in connected objects provide processing power and the ability to execute specific tasks or applications. In industrial domains, connected objects are often used for capturing properties such as temperature, and receiving signals or data from other devices. In domestic domains, many household products are expected to provide functions that could make human activities more convenient and improve the quality of life. For example, people can remotely control household equipment with smart electronics and embedded systems. To deploy IoT-based products and services, many IoT technologies have been utilized [25]:

- Radio Frequency Identification (RFID): identifies objects automatically and captures data using radio transmission, a tag and a reader.
- Wireless Sensor Networks (WSNs): measure, monitor and record the physical or environmental conditions.
- Middleware: makes communication and input/output between software applications easier for software developers.

[1]https://www.statista.com/statistics/471264/iot-number-of-connected-devices-worldwide

- Cloud computing: provides a infrastructure to handle and process the massive amounts of data generated by IoT.

The most typical networks of connected objects are Wireless Sensor Networks. It consists of autonomous sensor-equipped devices to monitor and sense the physical or environmental variables of our world [25, 28]. WSNs can be deployed with RFID systems to obtain more accurate measures, for instance, temperature, movements, and location [25, 5]. Sensor devices are connected by low energy wireless networks such as *Bluetooth Low Energy* (BLE)[2] and *IEEE 802.15.4*[3], and they send the streaming data to sink sensor nodes or heavier clients.

Streaming data is produced by connected objects in the IoT and transferred over the network. Different from an offline data-set, a data stream is a data model in which large volumes of data arrive continuously and cannot be saved completely [35]. Data points in a data stream can only be received in order, and it is impossible to randomly access the data [35].

With the rise of data science, data becomes increasingly important as it can provide knowledge and information after filtering and learning. Data streams are an important way to collect data and thus enable data science. The large volume and rapid velocity of data streams require that systems handle or save streaming data in a timely manner. We may lose the opportunity to process the data at all, if we do not do it in real time. This requirement creates algorithmic challenges, related to the restriction of memory and computing power in connected objects [35]. Some previous research has reviewed problems in streaming data analysis, such as finding frequent elements, estimating quantiles, or detecting patterns in data stream [21].

In general, there are two different types of sensor streams: (1) one-dimensional stream, for instance, temperature, humidity, and pressure, and (2) multi-dimensional streams, for instance 3D streams such as accelerometer, gyroscope, and magnetometer. One-dimensional streams are quite common in our daily life, and much research has been targeting them [22, 36, 52]. In contrast, multi-dimensional streams have not been widely studied in the literature, although they are equally popular.

In the field of IoT networks, power consumption is among the biggest challenges targeting connected objects, particularly in industrial domains, where several sensing

[2]https://web.archive.org/web/20170310111443/https://www.bluetooth.com/what-is-bluetooth-technology/how-it-works/low-energy

[3]http://www.ieee802.org/15/pub/TG4.html

systems are commonly launched in the field to run for days or even weeks without being recharged. Typically, such devices use sensors to capture properties such as temperature or motion, and stream them to a host system over a radio transmission protocol such as Bluetooth Low Energy (BLE). With the increasing computing requirements of embedded systems, connected objects that have a small battery capacity cannot operate for a long time while transmitting data. To extend the lifetime of objects without decreasing computing power, system designers aim to reduce the rate of data transmission as much as possible, as radio transmission is a power-hungry operation.

1.2 Goal of the thesis

Reducing the rate of data transmission is a good way to decrease the power consumption in connected objects. In this thesis, we intend to find a data summarization method able to shorten the length of the transmitted stream while retaining the important information in it. Furthermore, the method has to deal with the limitations of memory and processing power in connected objects and the multi-dimensional streams from sensors. In other words, this method must be non-resource-intensive, and it has to work for both one-dimensional and multi-dimensional streams. Summarization can be considered as a transformation of the original data to smaller summaries or patterns which contain as much information as possible. With data stream summarization, we might reduce the rate of transmitted data. For instance, only one data point will be transmitted rather than all data points, if all the elements in the stream are identical.

Data compression is a data summarization technique representing the data stream into a compacted version. Many data compression algorithms have been proposed for text compression [45, 42] and image compression [46, 53], but most previous algorithms are unable to fit into connected objects because of their limited computation power and memory. In addition, the compression process also costs energy. Thus, we also need to compare the energy cost of compression and the saved energy from reducing the rate of data transmitted. As more computation requires more energy [39], a compression algorithm with low computational complexity is needed. Meanwhile, the common way to handle multi-dimensional (n-dimensional) data streams is to compress each dimension independently, which boils down to compressing n one-dimensional data streams at the same time. But the parameters in multi-dimensional data points are dependent on each other, we have to consider them as a whole and process them together. Overall, we aim at finding a compression algorithm that adapts to multi-dimensional data streams and requires low computational and time complexity to reduce the size of transmitted data.

Lightweight Temporal Compression (LTC) [43] is one of stream compression method which has been designed specifically for energy-constrained systems, initially sensor networks. It approximates data points by a piece-wise linear function that guarantees an upper bound on the reconstruction error, and a reduced memory footprint in $\mathcal{O}(1)$. However, LTC has only been described for 1D streams, therefore, in this paper, we extend LTC to dimension n.

In our experiments, we test compression methods in Motsai's Neblina module, a system with a Nordic Semiconductor nRF52832 micro-controller, 64 KB of RAM, and Bluetooth Low Energy connectivity. Neblina has a 3D accelerometer, a 3D gyroscope, a 3D magnetometer, and environmental sensors for humidity, temperature, and pressure. The platform is equipped with sensor fusion algorithms for 3D orientation tracking and a machine learning engine for complex motion analysis and motion pattern recognition [41].

1.3 Outline and contributions

The contributions of this thesis are the following:

1. Formalize the description of original LTC algorithm.
2. Propose an algebraic formulation of n-dimensional LTC algorithm, and also introduce an norm-independent expression according to the formulation.
3. Implement LTC n-dimension for Infinity and Euclidean norms.
4. Validate the behavior of LTC n-dimension.
5. Measure the impact of LTC n-dimension on energy consumption.
6. Compare LTC n-dimension with Polynomial regression compression method.

Our implementation of LTC n-dimension is available as free software in https://github.com/big-data-lab-team/stream-summarization under MIT license, and it has already been implemented into Motsai's Neblina module[4] during my internship at Motsai from September 2018 to December 2018. Moreover, the extension of LTC is included in a data stream algorithm library named "OrpailleCC" available at https://github.com/big-data-lab-team/OrpailleCC and currently under review in the Journal of Open-Source Software.

In the rest of this thesis, Chapter 2 provides some background on stream summarization, lossless compression, and lossy compression methods. Chapter 3 formalizes the description of the LTC algorithm initially proposed in [43] and presents our norm-independent extension to dimension n and its implementation. Chapter 4 reports on experiments to validate our implementation, evaluates the impact of n-dimensional LTC on energy consumption of connected objects, and compares n-dimensional LTC with polynomial regression compression method.

The contents of Chapter 3 and Section 4.1 in Chapter 4 are included in our paper "A multi-dimensional extension of the Lightweight Temporal Compression method" [27], published in the 3rd Workshop on Real-time and Stream Analytics in Big Data & Stream Data Management, co-located with the IEEE Big Data conference 2018.

[4]https://motsai.com/products/neblina

Chapter 2

Related Work

2.1 Introduction

Streaming data has become increasingly important since the rise of the Internet of Things. Devices in the IoT communicate using streaming data, transmitted at high speed and short intervals, which may never be reviewed if the system does not process or store it immediately. To decrease the amount of data transmitted, stream summarization is useful and effective. Summarization can be considered as a process to discover a compressed description of the original data-set with the lowest possible *information loss* [8].

Some stream summarization techniques are used for data reduction. For instance, *Sampling* techniques, including uniform random sampling [51, 3], Reservoir sampling [50, 1] and weighted sampling [10, 14], capture a sub-sample of the data stream to represents the entire stream. *Filtering* techniques reduce the number of items in the stream: Bloom filter [6] and Cuckoo filter [15] are well-known methods and have been applied in many fields. In addition, to answer queries over a data stream, computing approximate result is more suitable for any queries than the exact solution [21]. Previous research has provided some synopsis constructions for summarization. *Histogram* techniques [21, 2] that give a distribution of items in the stream. *Sketch* techniques are able to solve some specific problem, for instance, Flajolet and Martin Sketch (FM-Sketch) [16, 17] can solve the problem of finding the number of distinct elements, Alon-Matias-Szegedy Sketch (AMS Sketch) [4] is able to estimate the second frequency moment, and Count-Min Sketch [11, 17] can calculate the quantiles and find frequent items. Moreover, many summarization methods apply *Sliding window* [13] technique which maintains a window that moves with new data coming. It ensures that the methods always use fresh data for analysis and statistics by keeping the most recent items of the stream or all items within a specific time period in given bound memory [26]. However, many summarization techniques only focus on specific problems and might eliminate most or partial data information. In some case, the integrity of data information and accuracy of query answers are required, thus we need a summarization technique which can both reduce the size of the stream and keep integral data information. Data compression is such a technique that meets the above conditions.

Data compression is a technique to reduce the number of bits required to represent a data set. It is also considered as a summarization technique that can give a compact version of the entire original data [19]. Data compression is categorized into *lossless* and *lossy* compression:

- *Lossless* compression methods remove statistical redundancy and the original data can be retrieved through decompression without any information loss [19].

- *Lossy* compression methods omit some information in the original data, but ensure that the reconstructed data has certain accuracy. For lossy compression, there is a trade-off between reconstruction accuracy and additional gains in terms of compression ratio [57].

In our case, the compression technique is the best choice of data summarization to reduce the rate of radio transmission, because losing any data points after reducing data stream is undesired, and compression guarantees the integrity of data stream. In other words, we can obtain the original data stream through the decompression process, thus no data points will be lost.

Most of the lossless compression methods belong to entropy or dictionary coding. Their main idea is to represent the new data points based on the "statistical model" or "dictionary", generated according to the data points we have seen. In general, the "statistical model" and "dictionary" help us to map the data points into bit sequences, thus compressing the data set. Huffman coding [20] and arithmetic coding [24] are the primary and classical entropy coding methods. Lossless Entropy Compression (LEC) algorithm [31] is a approximated version of exponential-Golomb code [49]. In

[31], LEC is utilized for compressing temperature and humidity streams by using a very small dictionary whose size is determined by the number of bits after Analog-to-Digital Converter (ADC) [31, 32]. The Sequential Lossless Entropy Compression (S-LEC) [28] algorithm is an extension and improvement of LEC algorithm. It exploits the positional relationship of groups of adjacent residues, in order to increase the compression ratio.

In dictionary-based lossless compression algorithms, Lempel-Ziv-77 (LZ77) [54] and Lempel-Ziv-78 (LZ78) [55] are well-known algorithms, where LZ77 maintains a sliding window as a dictionary during compression. There are several variations of LZ77 and LZ78, for instance, Lempel-Ziv-Welch (LZW) [40], Lempel-Ziv-Storer-Szymanski (LZSS) [48], Lempel-Ziv-Oberhumer (LZO) [33] etc. In order to suit these algorithms to connected objects and data streams, many improvements were proposed, such as Accelerometer-Lempel-Ziv-Storer-Szymanski (Accelerometer-LZSS, A-LZSS) [39] which combines LZSS and Huffman coding to compress accelerometer data, and Sensor-Lempel-Ziv-Welch (Sensor-LZW, S-LZW) [40], an algorithm which adapts LZW to sensor nodes.

Lossy compression methods are particularly suitable for sensor data streams, because measured sensor data intrinsically involves noise and measurement errors, which can be treated as a configurable tolerance for a lossy compression algorithm [27]. Thus, in this thesis, we only focus on lossy compression methods.

Resource-intensive lossy compression algorithms such as the ones based on polynomial interpolation, discrete cosine and Fourier transforms, or auto-regression methods [30] are not well-suited for connected objects, due to the limited memory available on these systems (typically a few KB), and the energy consumption associated with CPU usage [27]. Instead, compression algorithms need to find a trade-off between reducing network communications and increasing memory and CPU usage. As discussed in [57], linear compression methods provide a very good compromise between these two factors, leading to substantial energy reduction [27]. In this chapter, we will review several lossy compression methods in Section 2.2.

2.2 Lossy Compression

In this thesis, we only focus on the lossy compression method, because the measured sensor data intrinsically involves noise and measurement errors, which can be treated as a configurable tolerance for a lossy compression algorithm. Meanwhile, we sometimes prefer to sacrifice certain accuracy of reconstructed data for a better compression ratio. In our case, Neblina has limited memory, and in order to save more energy, we prefer a compression method with low computational complexity so that the energy cost of the compression process is not high. In this section, we list several lossy compression methods for streaming data.

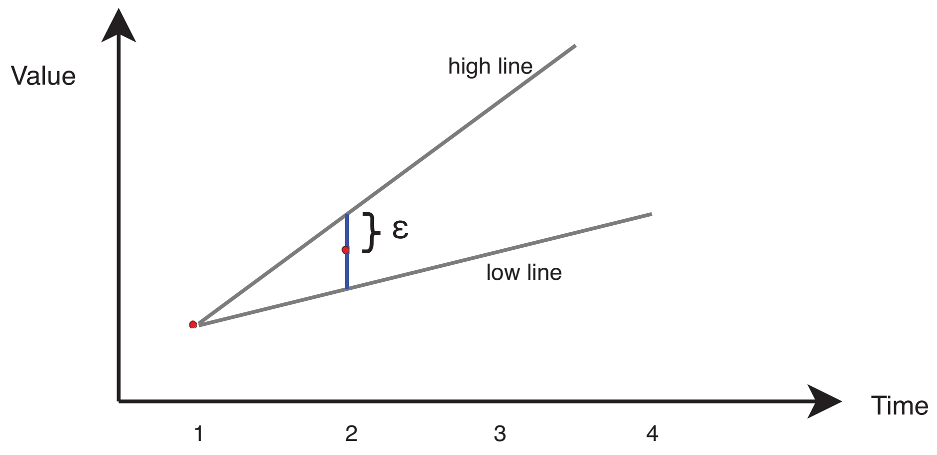
2.2.1 Lightweight Temporal Compression Algorithm

The Lightweight Temporal Compression (LTC) [43] algorithm approximates the data stream by a piece-wise linear function of time, with an error bounded by parameter ϵ.

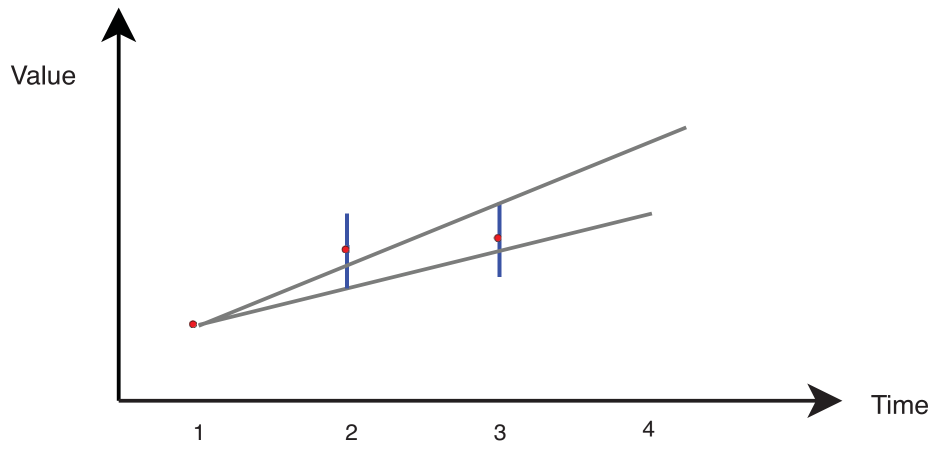
The LTC algorithm maintains two lines, the *high line*, and the *low line* defined by (1) the latest transmitted point and (2) the *high point* (high line) and the *low point* (low line). When a point (t_i, x_i) is received, the high line is updated as follows: if $x_i+\epsilon$ is below the high line then the high line is updated to the line defined by the last transmitted point and (t_i, $x_i + \epsilon$); otherwise, the high line is not updated. Likewise, the low line is updated from $x_i - \epsilon$. Therefore, any line located between the high line and the low line approximates the data points received since the last transmitted point with an error bounded by ϵ [43]. We assume that the points on the high line are (t_i, hp_i), and the points on low line are (t_i, lp_i), where hp_i and lp_i are the value of high line and low line at corresponding time t_i. The point (t_{i-1}, $\frac{hp_{(i-1)}+lp_{(i-1)}}{2}$) shall be transmitted if the received point meets the condition: $x_i + \epsilon < lp_i$ or $x_i - \epsilon > hp_i$. A example is presented in Figure 1. From Figure 1b, high line and low line are created and updated when we receive point at time t_2 and t_3, but the condition $x_4 + \epsilon < lp_4$ is met when point (t_4, x_4) come and we transmit point (t_3, $\frac{hp_3+lp_3}{2}$).

2.2.2 Piece-wise Linear Approximation with Minimum number of Line Segments Algorithm

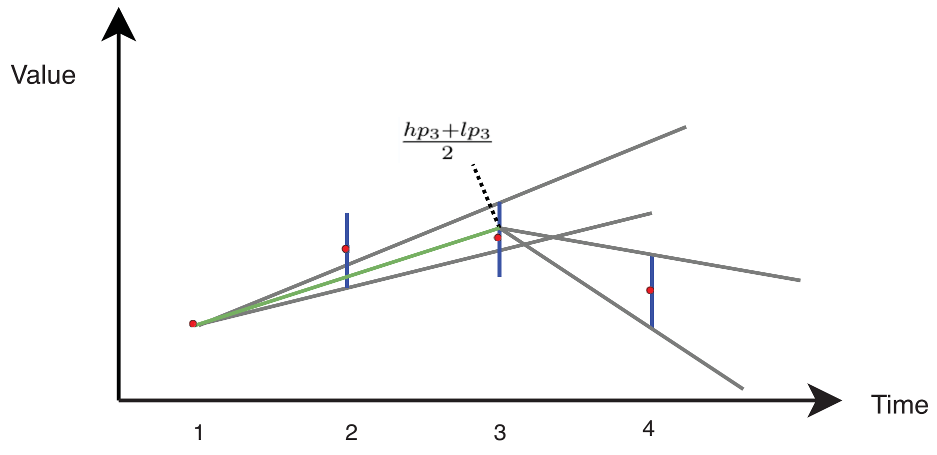
Similar to LTC, Piece-wise Linear Approximation with Minimum number of Line Segments (PLAMLis) [29] represents the original stream through a sequence of line



(a) Create *high line* and *low line*



(b) Update *high line* and *low line*



(c) Transmit data point

Figure 1: Lightweight temporal compression example

segments. The main idea of this algorithm is to find the minimum number of line segments to approximate the time series so that the amount of data transferred is reduced. Therefore, compressing stream is considered to be the problem "to represent stream data over a time window using a minimum number of segments". PLAMLis gives a greedy algorithm solution. Assume the input stream data points $X = \{x_1, ..., x_W\}$ are received over a time window of size W. Firstly, for each data points x_i, $i \in \{1, ..., W\}$, a longest segment S_i from point x_i to point x_j $(j > i)$ is built within the error bound. Thereby for the data points in the window, a sequence of longest segments $S = S_1, ..., S_W$ is obtained. Secondly, to pick the minimum number of subsets of S for representing original stream X, a greedy algorithm is used to select the segment S_k $(k \in [\![1, W]\!])$ which covers the largest number of data points x_i in X at each time, then remove it from S and add it into a `result sequence` until all data points in X are covered [29]. The result sequence is the result of compression [56, 57].

2.2.3 Enhanced Piece-wise Linear Approximation with Minimum number of Line Segments Algorithm

Enhanced Piece-wise Linear Approximation with Minimum number of Line Segments (Enhanced PLAMLis) [37] solves the problem "to represent stream data over a time window through using minimum number of segments" with a top-down recursive segmentation algorithm which has a smaller computational cost than PLAMLis [37, 57]. Assume W data points x_i in the time window, the segment $S_{(1,W)}$ with end points x_1 and x_W is created, then we have to check whether the maximum error is within error tolerance ϵ. If the maximum error is bigger than ϵ, the segment is split into two shorter segments $S_{(1,k)}$ and $S_{(k,W)}$ in data point x_k, $1 < k < W$. This procedure is applied recursively on each segment until the maximum error of all segments is within the error tolerance [37, 57].

2.2.4 Polynomial Regression

Different from piece-wise linear approximations, Polynomial Regression [57] gives a higher order $p \geqslant 1$ approximation of the data points by using standard regression methods based on least squares fitting [38]. The approximation is a sequence of curves (order $= p$) rather than linear segments. The algorithm starts with collecting $p+1$ samples $\{x_1, ..., x_{p+1}\}$ to obtain the coefficients of first p-order polynomial function. Upon receiving one sample x_{p+1+i} at each time, where x_{p+1+i} indicates the $(p+1+i)_{th}$ sample $(i > 0)$ in this approximation cycle, the best-fitting polynomial coefficients are re-computed with $\{x_1, ..., x_{p+1+i}\}$ and the algorithm checks whether the new polynomial approximates the data points within the desired error tolerance. If not, the coefficients of the previous regression are transmitted and a new approximation starts at the current sample [57].

During the compression process, all the points between transmissions need to be kept in memory, and the least squares fitting required larger computational cost than piece-wise linear approximations. However, polynomial regression gives better performance in terms of Root-Mean-Square Error (RMSE) between reconstructed data and original data. It means that the result from the regression method is closer to original data than result from the piece-wise linear approximation method [57].

2.2.5 Adaptive Auto-Regression Moving-Average technique

Adaptive Auto-Regression Moving-Average (A-ARMA) [30] is an improved version of Auto-Regression Moving-Average (ARMA). ARMA model is formed by combining AR and MA model, and it is usually used as a tool to predict future values over time series data [9]. The ARMA model `ARMA(`p`, `q`)` contains p AR terms and q MA terms. It is defined as:

$$X_t = \sum_{i=1}^{p} a_i X_{t-i} + Z_t + \sum_{i=1}^{q} \beta_i Z_{t-i} \tag{2.1}$$

The equation is reproduced from [9]. $a_1, ..., a_p$ and $\beta_1, ..., \beta_q$ are parameters of AR model and MA model respectively, $Z_{t-q}, ..., Z_t$ are white noise (is usually understood as residuals of the previous forecasts, $Z_t = X_t - X_{t-1}$) [9].

Similar to the ARMA model, A-ARMA is also composed of two terms, `AR` term and `MA` term, respectively predicting data value using $p(q)$ prior values or errors. To deal with the limit of computational complexity, A-ARMA adopts low-order ARMA with sliding window model [30]. The main idea of A-ARMA is maintaining and updating a ARMA model in memory based on sliding window.

Let's assume W is a sliding window with W window size, th_{err} is the minimum error tolerance on Root-Mean-Square Error (RMSE) and S means the length of each movement of sliding window. The adapted algorithm of A-ARMA is given in Algorithm 1. $model_{(p,q)}$ is the parameters of ARMA(p, q) model, obtained through function `build_ARMA()`. Function `go_forward()` makes the sliding window W to move S length (read S data samples), and function `tail(S)` returns S samples at the end of the window. RMSE is calculated by function `compute_error()`.

The first W data points are used to initialize the ARMA model, and to compare the RMSE between the original and predicted subsequent S data by moving sliding window S length each time. If the RMSE is larger than th_{err}, the saved ARMA model is remodeled with the current samples in sliding window [30]. In the decompression process, the stream data are predicted based on the parameters transmitted.

Algorithm 1 A-ARMA algorithm, adapted from [30]

```
1: Input
2:     $stream$          Data stream received
3:     $W$               Sliding window
4:     $th_{err}$        Threshold of error tolerance on root-mean-square error
5:     $S$               Length of sliding window move
6:     $p$               Order of AR term
7:     $q$               Order of MA term
8: Output
9:     $model_{(p,q)}$   Parameters of ARMA($p$, $q$) model
10: Read stream till $W$ is full                        ▷ Get first $W$ data from $stream$
11: $model_{(p,q)}$ = build_ARMA($W$.samples, $p$, $q$)        ▷ Build ARMA model
12: while $stream$ is not empty do
13:     $W$.go_forward($S$)                ▷ Moving sliding window forward $S$ length
14:     $samples = W$.tail($S$)
15:     $RMSE$ = compute_error($samples$, $model_{(p,q)}$.predict())
16:     if $RMSE > th_{err}$ then
17:         $model_{(p,q)}$ = build_ARMA($W$.samples, $p$, $q$)
18:         return $model_{(p,q)}$
19:     else
20:         return null                ▷ No transmitted data, model does not change
21:     end if
22: end while
```

2.2.6 Modified Adaptive Auto-Regression

Modified Adaptive Auto-Regression (MA-AR) is a modified version of A-ARMA, proposed by Zordan et al. [56]. In the A-ARMA algorithm, the ARMA model is built or updated over fixed window of W samples. It might cause bad performance to predict next S samples with the trained ARMA model over a fixed window, especially in highly noisy environments [56]. Assuming the prediction cycle means a process to find a AR model which represents as much original data as possible within error tolerance. The MA-AR algorithm uses a p-order AR model for each prediction cycle instead of sliding window, and controls the absolute error on each data rather than RMSE of S continuous data. Assume $M^{(n,i)}$ indicates the AR model built according to data $\{x_n, ..., x_{n+p-1+i}\}$, where $i > 0$, and $\hat{x}_{n+p-1+i}$ indicates the predicted data, then for each prediction cycle, MA-AR works as follows:

1. Collect first p samples in sensor node and send them to client side.
2. Collect one sample $x_{n+p-1+i}$ at a time, $i > 0$, to build p-order AR model $M^{(n,i)}$.
3. Predict $x_{n+p-1+j}$ where $j \in \{1, ..., i\}$ using $M^{(n,i)}$.
4. Check whether error $|\hat{x}_{n+p-1+j} - x_{n+p-1+j}|$ is larger than error tolerance ϵ.
    - If $|\hat{x}_{n+p-1+j} - x_{n+p-1+j}| \leqslant \epsilon$, the model is kept. Repeat from step 2.
    - Else the last model $M^{(n,i-1)}$ is encoded and transmitted, and new predict cycle is started from $x_{n+p-1+i}$.

The main idea of this algorithm is continuous estimations of the AR parameters. AR model is redefined only according to the last coming sample, so the computational cost is minimized and the parameters of the model can be computed through least squares minimization [56].

2.2.7 Comparison of compression algorithms

In [57], authors compare the performance of mentioned compression methods. They analyzed the performance in terms of compression ratio and energy consumption in the compression process for MA-AR (p=2, 4), Polynomial Regression (p=2, 4), PLAMLis, Enhanced PLAMLis and LTC. Regarding compression ratio, all the methods perform badly for small data sets, but as the data set size increases, the compression ratios of these methods increase until reaching their asymptote at 98%, 96%, 94%, and 90% respectively. In their experiment, polynomial regression gives the best compression ratio around 98%; and PLAMLis which reaches 96% compression ratio is the second-best; next, LTC and Enhanced PLAMLis have the same performance, providing 94% compression ratio, when the length of data is large; finally, MA-AR method has the worst compression ratio at 90%. In terms of energy consumption for compression, Polynomial Regression requires the most processing energy, MA-AR and PLAMLis also need significant processing energy. LTC uses less energy for compression, because LTC only compares the high point and low point with the data point received, and the computational complexity of each comparison process is constant.

2.3 Conclusion

In this chapter, general lossless and lossy compression methods were presented. The LTC compression method fully meets our requirements because of its low computational $\mathcal{O}(n)$ and space complexity $\mathcal{O}(1)$. However, a problem is that LTC compression method has only been described for 1D streams, while streams acquired by connected objects, such as acceleration or gyroscopic data, are often multi-dimensional [27]. In the next chapter, we extend LTC to dimension n and give implementation and a norm-independent expression of it. In Chapter 4, we test LTC n-dimension method on 3D acceleration streams.

Chapter 3

Extension of LTC to Dimension n and Implementation

In this chapter we introduce notations to describe LTC formally, we provide a norm-independent formulation of LTC in dimension n, and we describe its implementation. By n we refer to the dimension of the data points x_i. To handle time, LTC actually operates in dimension $n + 1$.

3.1 Preliminary comments

We note that the formulation of LTC in [43] relies on the intersection of *convex cones* in dimension $n{+}1$. For $n = 1$, it corresponds to the intersection of triangles, which can efficiently be computed by maintaining boundary lines, as detailed in Section 2.2.1. In higher dimension, however, cone intersections are not so straightforward to compute, due to the fact that the intersection between cones may not be a cone.

To address this issue, we formulate LTC as an intersection test between *n-balls*, that is, segments for $n = 1$, disks for $n = 2$, etc. N-balls are defined from the *norm* used in the vector space of data points. For $n = 1$, the choice of the norm does not really matter, as all p-norms and the infinity norm are identical. In dimension n, however, norm selection will be critical.

3.2 Algebraic formulation of LTC

3.2.1 Definitions

The algorithm receives a stream of data points x_i at times t_i ($i \in \mathbb{N}$), and it transmits a stream of data points ξ_i at times τ_i ($i \in \mathbb{N}$). To simplify the notations, we assume that:

$$\forall k \in \mathbb{N},\ \exists! i \in \mathbb{N}\ \tau_k = t_i \tag{3.2}$$

That is, transmission times coincide with reception times. We define the *shifted received points* as follows:

$$\forall k \in \mathbb{N}\ , \forall j \in \mathbb{N}^*,\ (u_j^k, y_j^k) = (t_{i+j}, x_{i+j}), \tag{3.3}$$

where i is such that $t_i = \tau_k$ and:

$$\forall k \in \mathbb{N},\ (u_0^k, y_0^k) = (\tau_k, \xi_k). \tag{3.4}$$

This definition is such that y_j^k is the j^{th} data point received after the k^{th} transmission and u_j^k is the corresponding time-stamp. Figure 2 illustrates the notations and algorithm.

Using these notations and details in Section 2.2.1, the original LTC algorithm can be written as in Algorithm 2. For readability, we assume that access to data points is blocking, i.e., the program will wait until the points are available. We also assume that the content of variable `tr` is transmitted after each assignment of this variable. Function `line`, omitted for brevity, returns the ordinate at abscissa x (1st argument) of the line defined by the points in its 2nd and 3rd arguments.

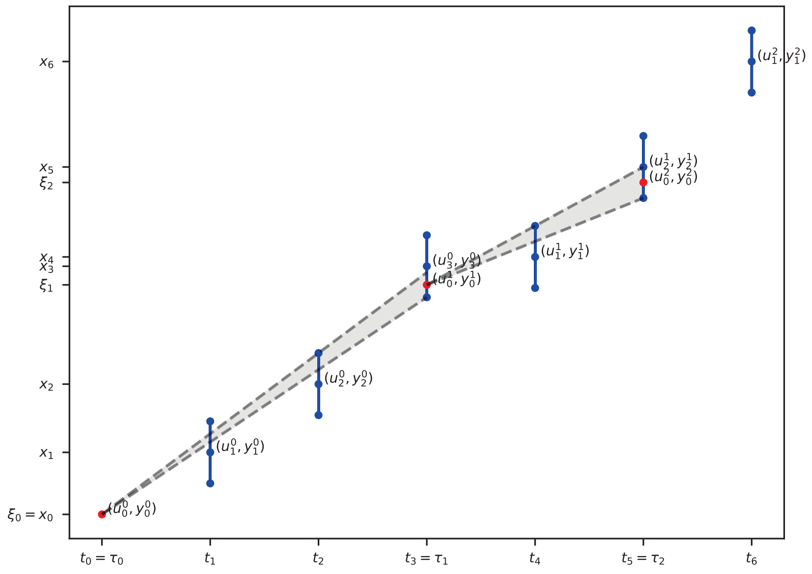


Figure 2: Illustration of the LTC algorithm. Blue dots are received points, red dots are transmitted points. Dashed lines represent the high and low lines when a point is transmitted.

According to equations (3.3) and (3.4), let $(u_0^k, y_0^k) \in \mathbb{R}^{n+1}$ be the latest transmitted point. For convenience, all the subsequent points will be expressed in the orthogonal space with origin (τ_k, ξ_k) through equation (3.4). We denote by $(v_j, z_j)_{j\in[\![0,m]\!]}$ such points:

$$\forall j \leq m,\ (v_j, z_j) = (u_j^k - \tau_k, y_j^k - \xi_k) \tag{3.5}$$

Let $\mathcal{B}_j$ be the ball of $\mathbb{R}^n$ of centre $\frac{v_1}{v_j} z_j$ and radius $\frac{v_1}{v_j}\epsilon$:

$$\mathcal{B}_j = \left\{ z \in \mathbb{R}^n, \left\| z - \frac{v_1}{v_j} z_j \right\| \leq \frac{v_1}{v_j}\epsilon \right\} \tag{3.6}$$

Note that v_1 is defined as soon as one point is received after the last transmission.

3.2.2 LTC property

We define the *LTC property* as follows:

$$\exists z \in \mathbb{R}^n,\ \forall j \in [\![1, m]\!], \left\| \frac{v_j}{v_1} z - z_j \right\| \leq \epsilon. \tag{3.7}$$

Algorithm 2 Original LTC algorithm, adapted from [43].

```
1: Input
2:     $(u_j^k, y_j^k)$   Received data stream
3:     $\epsilon$          Error bound
4: Output
5:     tr   Transmitted points
6: tr = $(u_0^0, y_0^0)$                                   ▷ Last transmitted point
7: k = 0 ; j = 1
8: (lp, hp) = $(y_1^0 - \epsilon,\ y_1^0 + \epsilon)$                 ▷ Low and high points
9: while True do                        ▷ Process received points as they come
10:     j += 1
11:     new_lp = max($y_j^k - \epsilon$, line($u_j^k$, tr, ($u_{j-1}^k$, lp)))
12:     new_hp = min($y_j^k + \epsilon$, line($u_j^k$, tr, ($u_{j-1}^k$, hp)))
13:     if new_lp ≤ new_hp then                              ▷ Keep compressing
14:         (lp, hp) = (new_lp, new_hp)
15:     else
16:         tr = $(u_{j-1}^k, (lp + hp)/2)$                          ▷ Transmit point
17:         k += 1
18:         j = 1
19:         (lp, hp) = $(y_j^k - \epsilon,\ y_j^k + \epsilon)$
20:     end if
21: end while
```

The original LTC algorithm ensures that the LTC property is verified between each transmissions. Indeed, all the data points z such that (v_1, z) is between the high line and the low line verify the property. Line 13 in Algorithm 2 guarantees that such a point exists.

The LTC property can be re-written as follows:

$$\exists z \in \mathbb{R}^n,\ \forall j \in [\![1, m]\!], \left\| z - \frac{v_1}{v_j} z_j \right\| \leq \frac{v_1}{v_j}\epsilon \tag{3.8}$$

that is:

$$\bigcap_{j=1}^{m} \mathcal{B}_j \neq \emptyset \tag{3.9}$$

Note that $(\mathcal{B}_j)_{j \in [\![1,m]\!]}$ is a sequence of n-balls of strictly decreasing radius, since $v_j > v_1$. The LTC algorithm generalized to dimension n tests that the LTC property in Equation 3.9 is verified after each reception of a data point. It is written in Algorithm 3.

Algorithm 3 Generalized LTC.

```
 1: Input
 2:     $(u_j^k, y_j^k)$   Received data stream
 3:     $\epsilon$          Error bound
 4: Output
 5:     tr   Transmitted points
 6: tr = $(\tau, \xi) = (u_0^0, y_0^0)$                    ▷ Last transmitted point
 7: k = 0 ; j = 0
 8: while True do
 9:     j += 1
10:     $(v_j, z_j) = (u_j^k - \tau, y_j^k - \xi)$
11:     if $\bigcap_{l=1}^{j} \mathcal{B}_l = \emptyset$ then
12:         Pick $z$ in $\bigcap_{l=1}^{j-1} \mathcal{B}_l$                ▷ Transmit point
13:         tr = $(\tau, \xi) = (u_{j-1}^k, z)$
14:         k += 1
15:         j = 1
16:     end if
17: end while
```

3.3 Intersections of n-balls

Although Algorithm 3 looks simple, one should not overlook the fact that there is no good general algorithm to test whether a set of n-balls intersect. The best general algorithm we could find so far relies on Helly's theorem which is formulated as follows [18]:

Theorem. *Let $\{X_i\}_{i \in [\![1,m]\!]}$ be a collection of convex subsets of $\mathbb{R}^n$. If the intersection of every $n+1$ subsets is non-empty, then the whole collection has an non-empty intersection.*

This theorem leads to an algorithm of complexity $\binom{m}{n+1}$ which is not usable in resource-constrained environments such as connected objects.

The only feasible algorithm that we found is norm-specific. It maintains a representation of the intersection $\bigcap_{j=1}^{m} \mathcal{B}_j$ that is updated at every iteration. The intersection tests can then be done in constant time. According to Equation (3.8) and Equation (3.9), the n-dimensional LTC algorithm can be used in all norms, such as Manhattan norm, Euclidean norm and Infinity norm. However, updating the representation of the intersection may be costly depending on the norm used. For the Infinity norm and Manhattan norm, the representation is a rectangular cuboid (horizontal or tilted) which is straightforward to update by intersecting with an n-ball. For the Euclidean norm, the representation is an arbitrary volume, which is more costly to maintain.

3.4 Effect of the norm

As mentioned before, norm selection in $\mathbb{R}^n$ results in different representations of intersection and has a critical impact on the compression error and ratio. To appreciate this effect, let us compare the Infinity norm and the Euclidean norm in dimension 2. By comparing the unit disk to a square of side 2, we obtain that the compression ratio of a random stream would be $\frac{4}{\pi}$ times larger with the infinity norm than with the Euclidean norm (see Figure 3). In 3D, this ratio would be $\frac{6}{\pi}$. Unsurprisingly, the infinity norm is more tolerant than the Euclidean norm.

It should also be noted that using the infinity norm in $\mathbb{R}^n$ boils down to the use of the 1D LTC algorithm independently in each dimension, since a data point will be

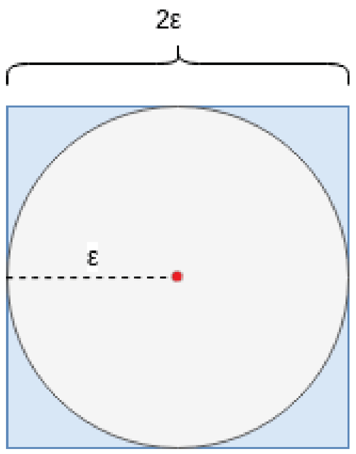


Figure 3: Comparison of compression ratio between Infinity norm and Euclidean norm in 2D

transmitted as soon as the linear approximation doesn't hold in any of the dimensions. For the other norms such as Euclidean norm and Manhattan norm, however, the multidimensional and multiple unidimensional versions are different: the multiple unidimensional version behaves as the infinity norm, but the multidimensional version is more stringent, leading to a reduced compression rate and error.

To choose between the multidimensional implementation and multiple unidimensional ones, we recommend to check whether the desired error bound is expressed independently for every sensor, or as an aggregate error between them. The multidimensional version is more appropriate for multidimensional sensors, for instance 3D accelerometers or 3D gyroscopes, and the multiple unidimensional version is more suitable for multiple independent sensors, for instance a temperature and a pressure sensor.

To compare the impact of norm selection on the compression error and ratio, in this thesis, we implement LTC in n dimensions with the Infinity norm and Euclidean norm corresponding to multiple unidimensional and multidimensional version respectively.

3.5 Implementation of LTC n-dimension

To implement LTC in n dimensions with the infinity norm, we maintain a cuboid representation of $\cap_{l=1}^{j}\mathcal{B}_l$ across the iterations of the `while` loop in Algorithm 3. The implementation works with constant memory and requires limited CPU time.

With the Euclidean norm, the intersection test is more complex. We keep in memory a growing set S of n-balls and the bounding box B which represents an approximate range of their intersection. We define B as follows:

$$B = \bigcap_{l=1}^{j-1} box(\mathcal{B}_l)$$

Where `box()` is a function that returns the bounding box of an n-ball. Then, when a new point arrives, we consider the associated n-ball $\mathcal{B}_j$ and our intersection test works as in Algorithm 4. Firstly, we check the intersection between box($\mathcal{B}_j$) and B (Algorithm 4, line 11). To restrict memory usage, B only covers a set S of n-balls were the size of S is limited to L_s. Secondly, for each $\mathcal{B}_i$ in S, we check the intersection between $\mathcal{B}_i$ and $\mathcal{B}_j$ (line 14). In addition $\mathcal{B}_i$ will be removed if it includes $\mathcal{B}_j$, because if $\mathcal{B}_j$ has intersection with n-balls in S then a bigger n-ball which contains $\mathcal{B}_j$ must also have intersection. Finally, we search a point in intersection of all n-balls in S, using plane sweep [44, 47] and bisection initialized by the bounds of B. Function `find_bisection(S, B)` (see Algorithm 5) and `recursive(S, L, R, N,` X^n`)` (see Algorithm 6) show how it works.

In function `find_bisection(S, B)`, we selected dimension of n-balls as n used at the beginning of bisection, also computed minimum/maximum value of bounding box B at the n_{th} dimension, which corresponds to $left$ and $right$ in bisection method. The return object X^n represents a n-dimensional point in the intersection of all n-balls in S. It would be returned if we have $True$ result from function `recursive(S, L, R, N,` X^n`)`. X^n is initialized and put into function `recursive(S, L, R, N,` X^n`)`, because it is necessary to update the values (see Algorithm 6 line 26) of X^n during the recursive process. In line 27 of Algorithm 6, it happens in case of $left > right$, which means:

$$mid \bigcap (min\_id \cap max\_id) = \emptyset$$

In Algorithm 4 line 14 and 15, it is guaranteed that any two n-balls in S has intersection. Therefore, any point in the intersection of min_id and max_id, could be used for determining the direction of bisection. In our implementation, we select a point in connection of these two n-ball's center, which also locates in intersecting object (line/chord when order=2, plane/circle when order=3). Figure 4 shows the point we selected in 2D.

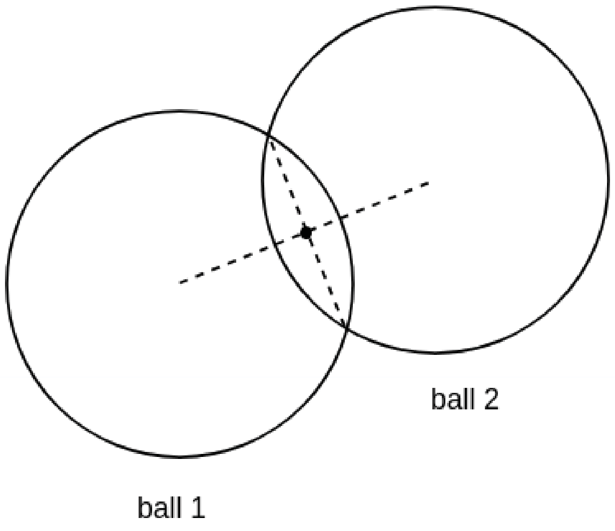


Figure 4: The point selected to determine the direction

According to our implementation for Euclidean norm, if the L_s is undefined (L_s is infinity), the operations in Algorithm 4 and Algorithm 5 before the function `recursive()` need $O(n)$ complexity. In the `recursive()` function, the bisection from line 12 to line 32 in Algorithm 6 needs $O(\log 2\epsilon)$, because the $left$ and $right$ are initialized from the bounding box B, where $|left - right| \leqslant 2\epsilon$. Moreover, computing $left$ and $right$ which used for next dimension in `recursive()` function needs $O(n)$. Thereby, when the data stream is two-dimensional, the time complexity of LTC n-dimension for Euclidean norm is $O(n) + O(\log 2\epsilon) \times O(n) = O(n \times \log 2\epsilon)$, where n is the number of the data points we seen so far. In the same way, if we process 3-dimensional stream with our algorithm with Euclidean norm, it needs extra $O(n \times \log 2\epsilon)$ time in recursion, the total time complexity is $O(n)+O(\log 2\epsilon)\times(O(n)+O(\log 2\epsilon)\times O(n)) = O(n\times(\log 2\epsilon)^2)$. Finally, our implementation for Euclidean norm gives $O(n \times (\log 2\epsilon)^{d-1})$ time complexity for d-dimensional data stream.

Algorithm 4 Intersection test for Euclidean n-balls.

```
 1: Input
 2:     $S$     Set of intersecting n-balls
 3:     $B$     Bounding box of the intersection of n-balls in $S$
 4:     $\mathcal{B}_j$    New n-ball to check
 5:     $L_s$    The maximum length of $S$
 6: Output
 7:     $S$     Updated set of intersecting n-balls
 8:     $B$     Updated bounding box
 9:     $T$     True if all the n-balls in S and $\mathcal{B}_j$ intersect
10:     $x$     Point inside intersection. It might be Null
11: if $S$.length $\geqslant L_s \parallel$ box($\mathcal{B}_j$) $\cap B = \emptyset$ then     ▷ N-ball is outside bounding box
12:     return ($S$, $B$, False, Null)
13: end if
14: if $\exists\, \mathcal{B}_i \in S$ s.t. $\mathcal{B}_j \cap \mathcal{B}_i = \emptyset$ then
15:     return ($S$, $B$, False, Null)                ▷ Some n-balls don't intersect
16: end if
17: if $\exists\, \mathcal{B}_i \in S$ s.t. $\mathcal{B}_j \subset \mathcal{B}_i$ then                    ▷ Remove inclusions
18:     Remove $\mathcal{B}_i$ from $S$.
19: end if
20: $B$ = box($\mathcal{B}_j$) $\bigcap$ $B$
21: $S = S \bigcup \{\mathcal{B}_j\}$
22: $x$ = find_bisection($S$, $B$)                      ▷ This can take some time
23: if $x$ == Null then
24:     return ($S$, $B$, False, Null)
25: else
26:     return ($S$, $B$, True, $x$)
27: end if
```

Algorithm 5 Function find_bisection(S, B)

```
1: Input
2:     $S$   Set of n-balls
3:     $B$   Bounding box of the intersection of n-balls in $S$
4: Output
5:     $X^n$   N-dimensional point in all n-balls in $S$, $X^n = (x_1, ..., x_n)$
6: $n$ = dimension of n-balls
7: $left$ = min_value($B$, $n$)          ▷ Minimum value of $B$ along $n^{th}$ dimension
8: $right$ = max_value($B$, $n$)         ▷ Maximum value of $B$ along $n^{th}$ dimension
9: $X^n$ = Null
10: if recursive(S, left, right, n, $X^n$) then
11:     return $X^n$
12: else
13:     return Null
14: end if
```

3.6 Conclusion

In this chapter, we provided a formulation for the original LTC algorithm and a pseudo-code with our notation and definitions. To generalize LTC to dimension n, the LTC property has been re-written in Section 3.2.2. According to Section 3.4, we know that the n-dimensional LTC might get different results when different norms are used. With the Euclidean norm, it is more difficult to check the intersection and record this intersection in memory. We introduce the method of combination of plane sweep and bisection for intersection test in Euclidean norm, but it spends more processing time and costs more memory to record a sequence of n-balls than the method for the Infinity norm. In next chapter, we will evaluate the performance of n-dimensional LTC on accelerometer data for human activities.

Algorithm 6 Function recursive(S, L, R, N, X^n)

```
1: Input
2:     $S$    Set of n-balls
3:     $L$    Min value used in bisection
4:     $R$    Max value used in bisection
5:     $N$    The $N^{th}$ dimension, $N \in \{1...n\}$
6:     $X^n$   N-dimensional point which records point of intersection, $X^n = (x_1, ..., x_n)$
7: Output
8:     $T$    True if a point is found by using bisection
9: if $N == 1$ then
10:     $X^n.x_N = \frac{L+R}{2}$;  return True
11: end if
12: while $L < R$ do
13:     $mid = \frac{L+R}{2}$;  $surface = \{x_N...x_n\}$
14:     $left$ = -$\infty$;  $right = +\infty$
15:     for all $\mathcal{B}_i \in S$ do
16:         $min$ = min_value($\mathcal{B}_i \cap surface$, N-1)
17:         $max$ = max_value($\mathcal{B}_i \cap surface$, N-1)
18:         if $left < min$ then
19:             $left = min$;  $min\_id = \mathcal{B}_i$
20:         end if
21:         if $right > max$ then
22:             $right = max$;  $max\_id = \mathcal{B}_i$
23:         end if
24:     end for
25:     if $left \leqslant right$ then
26:         $X^n.x_N = mid$;   return recursive($S$, $left$, $right$, $N-1$, $X^n$)
27:     else if $\forall$ $P$ (Point) $\in (min\_id \cap max\_id)$ s.t. $P.x_N < mid$ then
28:         $R = mid$
29:     else
30:         $L = mid$
31:     end if
32: end while
33: return False
```

Chapter 4

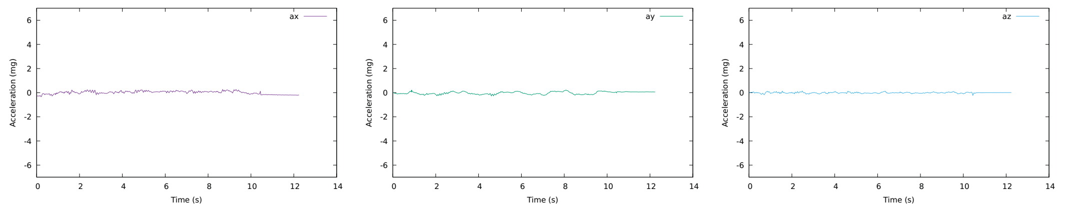
Experiments and Results

In the experiments, we first evaluate the LTC n-dimension with biceps curls data-set, and test the effect on energy consumption when it is used in Neblina. Then we compare LTC n-dimension and polynomial regression compression method in terms of the compression ratio in different dimensions. In order to restrict the memory usage of LTC n-dimension for the Euclidean norm, all the experiments in this chapter define that the maximum number L_s of n-balls saved in memory is 200.

The code and data-set used in experiments are provided on `https://github.com/big-data-lab-team/stream-summarization`.

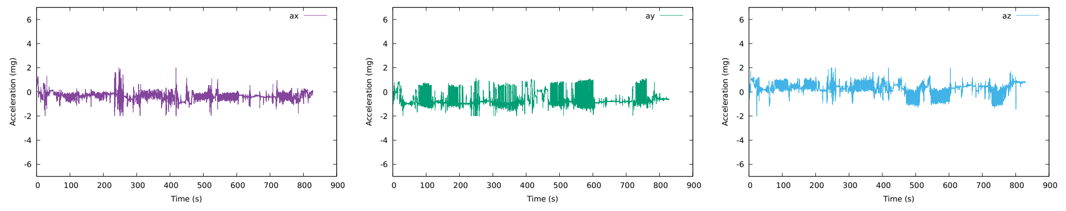
4.1 Evaluation of LTC n-dimension

We conducted two experiments using Motsai's Neblina module, a system with a Nordic Semiconductor nRF52832 micro-controller, 64 KB of RAM, and Bluetooth Low Energy connectivity. Neblina has a 3D accelerometer, a 3D gyroscope, a 3D magnetometer, and environmental sensors for humidity, temperature and pressure. The platform is equipped with sensor fusion algorithms for 3D orientation tracking and a machine learning engine for complex motion analysis and motion pattern recognition [41]. Neblina has a battery of 100mAh; at 200 Hz, its average consumption is 2.52 mA when using accelerometer and gyroscope sensors but without radio transmission, and 3.47 mA with radio transmission, leading to an autonomy of 39.7 h without transmission and 28.8 h with transmission.



(a) Short biceps curl[a]

[a] Average of az data is -7.81mg. It was shifted to 0 so that the graphs can all use the same y scale.



(b) Long biceps curl

Figure 5: Time-series used in validation

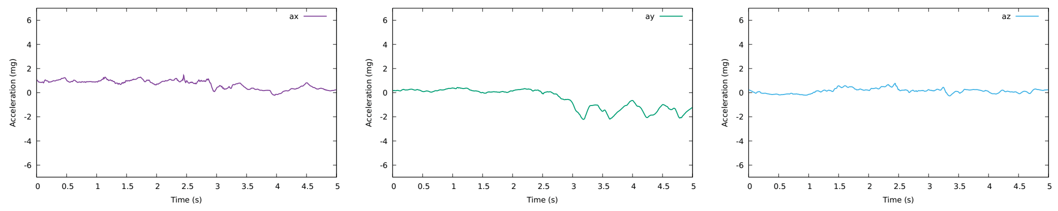
4.1.1 Experiment 1: validation

We validated the behavior of our LTC extension on a PC using data acquired with Neblina. We collected two 3D accelerometer time-series, a short one and a longer one, acquired on two different subjects performing biceps curl, with a 50 Hz sampling rate (see Figure 5). In both cases, the subject was wearing Neblina on their wrist. It should be noted that the longest time-series also has a higher amplitude, perhaps due to differences between subjects.

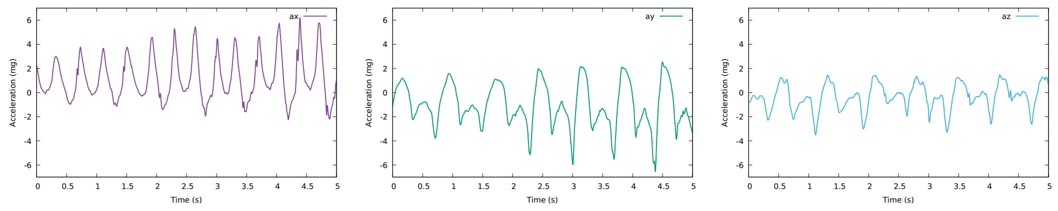
We compressed the time-series with various values of ϵ, using our 2D (x and y) and 3D (x, y and z) implementations of LTC. On Neblina, the raw uncalibrated accelerometer data corresponds to errors around 20 mg (1 g is 9.8 m/s^2). We used a laptop computer with 16 GB of RAM, an Intel i5-3210M CPU @ 2.50GHz $\times$ 4, and Linux Fedora 27. We measured memory consumption using Valgrind's massif tool [34], and processing time using `gettimeofday()` from the GNU C Library.

Results are reported in Table 1. As expected, the compression ratio increases with ϵ, and the maximum measured error remains lower than ϵ in all cases. The maximum is reached most of the time on these time-series.

Infinity vs Euclidean norms The average ratio between the compression ratios obtained with the infinity and Euclidean norms is 1.03 for 2D data, and 1.06 for 3D



(a) Walking



(b) Running

Figure 6: Time-series used for measuring energy savings

data. These ratios are lower than the theoretical values of $\frac{4}{\pi}$ in 2D and $\frac{6}{\pi}$ in 3D, which are obtained for random-uniform signals. Unsurprisingly, the infinity norm surpasses the Euclidean norm in terms of resource consumption. Memory-wise, the infinity norm requires a constant amount of 80 B, used to store the intersection of n-balls. The Euclidean norm, however, uses up to 4.7 KB of memory for the Long time-series in 3D with ϵ=48.8 mg. More importantly, the amount of required memory increases for longer time-series, and it also increases with larger values of ϵ. Similar observations are made for the processing time, with values ranging from 0.4 ms for the simplest time-series and smallest ϵ, to 41.3 ms for the most complex time-series and largest ϵ.

2D vs 3D For a given ϵ, the compression ratios are always higher in 2D than in 3D. It makes sense since the probability for the signal to deviate from a straight line approximation is higher in 3D than it is in 2D. Besides, resource consumption is higher in 3D than in 2D: for the infinity norm, 3D consumes 1.4 times more memory than 2D (1.8 times on average for Euclidean norm), and the processing time is 1.35 longer (1.34 on average for Euclidean norm).

	Infinity		Euclidean	
ϵ (mg)	48.8	34.5	48.8	34.5
Max error (mg)	48.8	34.4	48.8	34.5
Compression ratio (%)	79.77	72.59	77.49	70.96
Peak memory (B)	80	80	688	688
Processing time (ms)	0.101	0.094	0.456	0.406

(a) Short biceps curl (2D)

	Infinity		Euclidean	
ϵ (mg)	48.8	34.5	48.8	34.5
Max error (mg)	48.8	34.5	48.8	34.5
Compression ratio (%)	77.46	70.98	75.77	68.81
Peak memory (B)	80	80	2512	2608
Processing time (ms)	6.06	5.84	33.84	31.07

(b) Long biceps curl (2D)

	Infinity		Euclidean	
ϵ (mg)	48.8	28.2	48.8	28.2
Max error (mg)	48.8	28.2	48.8	28.2
Compression ratio (%)	78.14	66.39	74.39	63.13
Peak memory (B)	112	112	1744	784
Processing time (ms)	0.147	0.134	0.731	0.514

(c) Short biceps curl (3D)

	Infinity		Euclidean	
ϵ (mg)	48.8	28.2	48.8	28.2
Max error (mg)	48.8	28.2	48.8	28.2
Compression ratio (%)	71.23	58.11	67.35	53.24
Peak memory (B)	112	112	4752	3856
Processing time (ms)	7.87	7.22	41.29	39.04

(d) Long biceps curl (3D)

Table 1: Results of validation

4.1.2 Experiment 2: impact on energy consumption

We acquired two 3D accelerometer time-series at 200 Hz for two activities: walking and running (see Figure 6). In both cases, the subject was wearing Neblina on their wrist as in Experiment 1. We collected 1,000 data points for each activity, corresponding to 5 seconds of activity.

We measured energy consumption associated with the transmission of these time-series by "replaying" the time-series after loading them as a byte array in Neblina. We measured the current every 500 ms. We also measured the max and average latency resulting from compression.

Results are reported in Table 2. For a given ϵ and norm, the compression ratio is larger for walking than for running. The ratio of saved energy is relative to the reference current of 3.47 mA measured when Neblina transmits data without compression. In all cases, activating compression saves energy. The reduction in energy consumption behaves as the compression ratio: it increases with ϵ, it is higher for the infinity norm than for the Euclidean, and it is higher for the walking activity than for running. For a realistic error of ϵ=9.8 mg, the ratio of saved energy with the infinity norm is close to 20% for the walking activity, which is substantial. Latency is higher for walking than for running, and it is also higher for the Euclidean norm than for the infinity norm. In all cases, the latency remains lower than the 5-ms tolerable latency at 200 Hz, which demonstrates the feasibility of 3D LTC compression.

4.2 Comparison with Polynomial Regression

From the previous experiments, we know that LTC has good performance on walking and running accelerometer data sets. However as we have seen, the running data set (Figure 6) looks like it is made of many high-degree polynomials. We wonder whether high-degree regression compression method performs better with our accelerometer data sets. In this section, we implement the Polynomial Regression compression method mentioned in 2.2.4 and we compare it with LTC in different dimensions.

	Infinity			Euclidean		
ϵ (mg)	48.8	9.8	4.9	48.8	9.8	4.9
Max error (mg)	48.8	9.8	4.9	48.8	9.8	4.9
Compr. ratio (%)	88.9	66.4	45.5	87.6	63.3	37.2
Average (mA)	2.64	2.79	3.02	3.10	3.02	3.13
Saved energy (%)	23.9	19.7	13.0	10.7	13.0	9.7
Max latency (μs)	60	–	–	1530	–	–
Average latency (μs)	31	–	–	145	–	–

(a) Walking

	Infinity			Euclidean		
ϵ (mg)	48.8	9.8	4.9	48.8	9.8	4.9
Max error (mg)	48.8	9.8	4.9	48.8	9.8	4.9
Compr. ratio (%)	68.6	25.5	9.5	64.4	19.8	5.7
Average (mA)	2.88	3.22	3.38	2.95	3.32	3.39
Saved energy (%)	17.0	7.2	2.5	14.9	4.3	2.2
Max latency (μs)	60	–	–	840	–	–
Average latency (μs)	30	–	–	64	–	–

(b) Running

Table 2: Results of energy savings

4.2.1 Implementation of Polynomial Regression

We implemented polynomial regression method to compress 3D acceleration data (t, $a.x$, $a.y$, $a.z$). In the implementation, we process polynomial regression compression method on each accelerometer parameter. There are three polynomial functions between independent variables t (time-stamp) and dependent variables $a.x$, $a.y$ or $a.z$ (e.g. $t \sim a.x$). In other words, it means that three regression models shall be transmitted to represent a curve in 3D space. For tolerance checking, we use the infinity norm and Euclidean norm.

As we mentioned in Section 2.2.4, M_j^k means the j^{th} polynomial regression model after k^{th} transmission, but to apply this method on the acceleration data which is 3-dimensional, we need three models for it, one for each parameter. Assuming Mx_j^k, My_j^k and Mz_j^k represent the regression model on each parameter x, y and z respectively, Algorithm 7 shows how polynomial regression is applied on 3D acceleration data. In line 13, the coefficients of three regression models of parameters x, y and z are calculated, and each time a new point arrives they will be recalculated. The function `max_residue()` returns the maximum error between predicted data and original data. If the maximum residue is larger than ϵ, then the last regression model that meets error tolerance and the time-stamp of the last data are transmitted as the compression result.

We use the implemented polynomial regression method on the previous walking and running data sets. In the compression process, we need to keep in memory all the data between two transmissions and update regression models each time a new data point arrives.

4.2.2 Results

From Table 3a and Table 3b, we observe that LTC n-dimension gives better compression ratio than regression method on different dimensions, even though regression method represents original data with fewer segments (curve segments). It is because polynomial regression method requires more bytes to transmit the coefficients while LTC n-dimension just transmits a data point each time. In other words, we need to represent each regression curves with $(N+1) \times M \times 4 + 4$ bytes, but each straight line can be defined by two data points with $2 \times (M \times 2 + 4)$ bytes (2 bytes for each parameter). Assume Polynomial regression and LTC n-dimension needs G_P

Algorithm 7 Polynomial Regression Algorithm for 3D Accelerometer data

```
1: Input
2:     $(\chi, t_i)$ Received data stream at time $t_i$
3:     $\epsilon$ Error bound
4:     $p$ The order of polynomial function
5: Output
6:     tr Transmitted coefficients
7: $S = \emptyset$
8: $k$=0; $j$=0
9: while True do
10:     $S = S \cup (\chi, t_i)$
11:     if $S.length() \geqslant p + 1$ then
12:         j += 1
13:         $(Mx_j^k, My_j^k, Mz_j^k)$ = model($S$, $p$)    ▷ Compute coefficients
14:         if max_residue($Mx_j^k$, $My_j^k$, $Mz_j^k$, $S$) $> \epsilon$ then
15:             tr = $(Mx_{j-1}^k, My_{j-1}^k, Mz_{j-1}^k, t_{i-1})$
16:             k += 1; j = 0
17:             $S = \chi$
18:         end if
19:     end if
20: end while
```

curves and G_L lines respectively to represent original data set. Then we have to transmit $((N + 1) \times M \times 4 + 4) \times G_P$ bytes by using polynomial regression, and $2 \times (M \times 2 + 4) \times (G_L - 1)$ bytes ($G_L - 1$ data points) by using LTC n-dimension because LTC n-dimension generates connected straight lines. In our case, the polynomial regression method uses more bytes to represent original stream than LTC n-dimension. In reality, the compression ratio of these two methods depends on the data set, the polynomial regression method would give us higher compression ratio than LTC n-dimension, if the regression model really fits our data set.

Besides, similarly to Experiment 2 in Section 4.1.2, the compression ratio is higher for the infinity norm than for the Euclidean and is higher for the walking data set than for running data set. For the given degree of polynomial regression method, the higher degree results in a smaller compression ratio for walking and running in general, but the variations are slight. However, in Table 3a, when the dimension is 2 and 3, the 5-degree regression compresses more data than the 3-degree regression method in walking data set for Infinity norm. In Table 3b, the 5-degree regression method has a higher compression ratio than 3-degree regression for both norms when dimension equals 1 and for infinity norm when dimension equals 2.

Polynomial regression method needs $O(n)$ memory to save all the data points between two transmissions and long processing time to calculate coefficients and build model when each new data point comes. Because we implemented polynomial regression method and LTC n-dimension in different platforms, the comparison of memory usage and processing time between them cannot be measured.

4.3 Conclusion

This chapter demonstrated the effect of n-dimensional LTC. In the first section, we compressed biceps curl, walking and running steam data in different dimensions by using n-dimensional LTC. With error bound ϵ=48.8 mg, 3-dimensional LTC can compress at least 67% of original data stream. In section 4.1.2, we deployed LTC n-dimension on Neblina to measure the impact on energy consumption. We processed the experiment on Neblina with 3D accelerometer streams for walking and running activities. According to the Table 2, LTC n-dimension can reduce data transmission up to 88.9%, and help Neblina to save energy up to 23.9%, where the error bound

	LTC n-dimension		Regression degree = 3		Regression degree = 5	
Dimension	Infinity	Euclidean	Infinity	Euclidean	Infinity	Euclidean
1	90.1%	90.1%	89%	89%	88.96%	88.96%
2	89.6%	88.7%	83.35%	83.35%	83.55%	83.1%
3	88.9%	87.6%	80.24%	78.68%	80.8%	77.96%

(a) Comparison of compression ratios on Walking data set

	LTC n-dimension		Regression degree = 3		Regression degree = 5	
Dimension	Infinity	Euclidean	Infinity	Euclidean	Infinity	Euclidean
1	74.7%	74.7%	70.7%	70.7%	71.1%	71.1%
2	70.6%	68.6%	58.3%	57.4%	58.4%	57.35%
3	68.6%	64.4%	51.1%	48%	50.2%	47.92%

(b) Comparison of compression ratios on Running data set

Table 3: Results of Comparison

ϵ=48.8 mg. These experiments and results show that n-dimensional LTC is feasible on connected objects and it has good performance for compressing stream data to reduce energy consumption of transmission in IoT networks.

Chapter 5

Conclusion

5.1 Summary

IoT has been widely popularized in recent years and generates large amounts of streaming data. However, hardware limitations of connected devices create challenges: connected devices lack enough memory to store streaming data, micro-controllers can hardly run methods that have high computational complexity, and battery-based devices are limited in lifetime. Energy consumption is an urgent problem in IoT, especially in battery-powered devices, as computing and network transmission drain the battery quickly.

In this thesis, we aimed to find a summarization method which can be fitted in connected objects and can process multi-dimensional streams to reduce the number of data transmitted. Data compression is one of the techniques in data summarization, representing original data by a compact version and keeping all information (lossless compression) or retaining information at certain accuracy (lossy compression). We introduced a few general compression methods in this thesis and we finally selected the Lightweight Temporal Compression (LTC) algorithm. LTC is a lossy compression method that approximates data streams by a piece-wise linear function of time. It guarantees that the reconstruction error remains lower than a user-assigned error bound ϵ.

However, the original LTC method was only available for 1D data streams. Thus, we provided a formulation of LTC in dimension n, and the corresponding implementation. LTC can work with different norms in dimension n, but its compression result and costs depend on the norm used. In the experiments, we collected 2 biceps curl data-sets of different sizes. By using these data sets, we validate the performance of LTC n-dimension for Infinity and Euclidean norms in respect of compression ratio, memory and processing time. Then, we implemented the LTC n-dimension algorithm and deployed it on Neblina, the connected device platform developed by our Motsai partner. To measure the compression ratio and energy consumption of LTC in dimension n, we processed a experiment on Neblina with a dataset of human walk and run. Results showed that our n-dimensional implementation of LTC works on Neblina. By executing our method, Neblina can save at least 10% of energy when the error bound ϵ is 48.8 mg. Additionally, we compared LTC in dimension n to polynomial regression in different dimensions. We found that LTC in dimension n for Infinity and Euclidean norms outperform polynomial regression method in terms of compression ratio, regardless of the dimension or degree of regression. Our extension of LTC in dimension n has been deployed in Motsai's Neblina during my 4-month internship from September to December 2018.

5.2 Limitations

The main limitation of our current LTC implementation in dimension n is memory usage. Our formulation of LTC in dimension n boils down to an intersection test between n-balls, however, with the Euclidean norm this test is difficult. The Helly's theorem is the best algorithm we could find for this purpose, but it is still too complex for resource-constrained environments. We provided a method that utilizes plane sweep and bisection to determine if a set of n-balls intersect.

Our method has a $O(n \times (\log 2\epsilon)^{d-1})$ time complexity, where n is the number of data points we have seen so far since the last transmission, ϵ is the error tolerance and d is the dimension of the stream, but it requires extra memory to save n-balls between two transmissions. Even though we bound n to 200, it still costs 4.6 KB to compress the long bicep curl data set in our first experiment (see Table 1d).

In addition, compressing data in connected devices causes transmission latency since it needs processing time. In our case, we have to make sure the maximum latency is smaller than the sampling rate of sensors, because the process of current data point must be finished before receiving new data point. For instance, in Section 4.1.2, the sampling rate of accelerometer sensor is 200 Hz which means a new data point comes each 5-ms, thus the maximum latency must lower than 5-ms. From Table 2, when sampling rate is 200Hz, the latency with our method is lower than the 5-ms tolerable latency. But LTC n-dimension for Euclidean norm cannot work in high sampling rate, such as 800Hz, corresponding 1.25-ms tolerable latency.

A better algorithm would be needed to deal with the memory usage and processing time of the intersection test with the Euclidean norm.

5.3 Future work

In our future work, we would like to find algorithms that have lower time and space complexity to handle the intersection check. We will review other compression algorithms, attempting to achieve more energy savings. Moreover, lossless algorithms are necessary sometimes, typically in e-health applications. A framework which can support both efficient lossless and lossy algorithm and can work with the multi-dimension stream is also one of the research direction in our future work.

Bibliography

[1] Charu C. Aggarwal. Data streams - models and algorithms. In *Advances in Database Systems*, 2007.

[2] Mohiuddin Ahmed. Data summarization: a survey. *Knowledge and Information Systems*, 58(2):249–273, 2019.

[3] Joachim H. Ahrens and Ulrich Dieter. Sequential random sampling. *ACM Trans. Math. Softw.*, 11:157–169, 1985.

[4] Noga Alon, Yossi Matias, and Mario Szegedy. The space complexity of approximating the frequency moments. *Journal of Computer and system sciences*, 58(1):137–147, 1999.

[5] L Atzori and Antonio Iera. b., giacomo morabito, c.: The internet of things: a survey. *Comput. Netw*, 2010.

[6] Burton H Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426, 1970.

[7] Nadine Boudargham, Jacques Bou Abdo, Jacques Demerjian, and Christophe Guyeux. Exhaustive study on medical sensors. In *SENSORCOMM 2017, The Eleventh International Conference on Sensor Technologies and Applications*, pages 86–93. IARIA, 2017.

[8] Varun Chandola and Vipin Kumar. Summarization–compressing data into an informative representation. *Knowledge and Information Systems*, 12(3):355–378, 2007.

[9] Chris Chatfield. *The analysis of time series: an introduction.* Chapman and Hall/CRC, 2016.

[10] Surajit Chaudhuri, Rajeev Motwani, and Vivek Narasayya. On random sampling over joins. In *ACM SIGMOD Record*, volume 28, pages 263–274. ACM, 1999.

[11] Graham Cormode and Shan Muthukrishnan. An improved data stream summary: the count-min sketch and its applications. *Journal of Algorithms*, 55(1):58–75, 2005.

[12] Li Da Xu, Wu He, and Shancang Li. Internet of things in industries: A survey. *IEEE Transactions on industrial informatics*, 10(4):2233–2243, 2014.

[13] Mayur Datar, Aristides Gionis, Piotr Indyk, and Rajeev Motwani. Maintaining stream statistics over sliding windows. *SIAM journal on computing*, 31(6):1794–1813, 2002.

[14] Pavlos S Efraimidis and Paul G Spirakis. Weighted random sampling with a reservoir. *Information Processing Letters*, 97(5):181–185, 2006.

[15] Bin Fan, Dave G Andersen, Michael Kaminsky, and Michael D Mitzenmacher. Cuckoo filter: Practically better than bloom. In *Proceedings of the 10th ACM International on Conference on emerging Networking Experiments and Technologies*, pages 75–88. ACM, 2014.

[16] Philippe Flajolet and G Nigel Martin. Probabilistic counting algorithms for data base applications. *Journal of computer and system sciences*, 31(2):182–209, 1985.

[17] Minos Garofalakis, Johannes Gehrke, and Rajeev Rastogi. *Data Stream Management: Processing High-Speed Data Streams*. Springer, 2016.

[18] Ed Helly. Über mengen konvexer körper mit gemeinschaftlichen punkte. *Jahresbericht der Deutschen Mathematiker-Vereinigung*, 32:175–176, 1923.

[19] ZR Hesabi, Zahir Tari, A Goscinski, Adil Fahad, Ibrahim Khalil, and Carlos Queiroz. Data summarization techniques for big dataa survey. In *Handbook on Data Centers*, pages 1109–1152. Springer, 2015.

[20] David A Huffman. A method for the construction of minimum-redundancy codes. *Proceedings of the IRE*, 40(9):1098–1101, 1952.

[21] Arun Kejariwal, Sanjeev Kulkarni, and Karthik Ramasamy. Real time analytics: algorithms and systems. *Proceedings of the VLDB Endowment*, 8(12):2040–2041, 2015.

[22] Bernard M Kulwicki. Humidity sensors. *Journal of the American Ceramic Society*, 74(4):697–708, 1991.

[23] Xiaochen Lai, Quanli Liu, Xin Wei, Wei Wang, Guoqiao Zhou, and Guangyi Han. A survey of body sensor networks. *Sensors*, 13(5):5406–5447, 2013.

[24] Glen G Langdon. An introduction to arithmetic coding. *IBM Journal of Research and Development*, 28(2):135–149, 1984.

[25] In Lee and Kyoochun Lee. The internet of things (iot): Applications, investments, and challenges for enterprises. *Business Horizons*, 58(4):431–440, 2015.

[26] Jure Leskovec, Anand Rajaraman, and Jeffrey David Ullman. *Mining of massive datasets*. Cambridge university press, 2014.

[27] Bo Li, Omid Sarbishei, Hosein Nourani, and Tristan Glatard. A multi-dimensional extension of the lightweight temporal compression method. In *3rd Workshop on Real-time and Stream Analytics in Big Data & Stream Data Management, co-located with the IEEE International Conference on Big Data*, pages 2918–2923. IEEE, 2018.

[28] Yimei Li and Yao Liang. Temporal lossless and lossy compression in wireless sensor networks. *ACM Transactions on Sensor Networks (TOSN)*, 12(4):37, 2016.

[29] Chong Liu, Kui Wu, and Jian Pei. An energy-efficient data collection framework for wireless sensor networks by exploiting spatiotemporal correlation. *IEEE transactions on parallel and distributed systems*, 18(7):1010–1023, 2007.

[30] Jialiang Lu, Fabrice Valois, Mischa Dohler, and Min-You Wu. Optimized data aggregation in wsns using adaptive arma. In *2010 Fourth International Conference on Sensor Technologies and Applications*, pages 115–120. IEEE, 2010.

[31] Francesco Marcelloni and Massimo Vecchio. A simple algorithm for data compression in wireless sensor networks. *IEEE communications letters*, 12(6):411–413, 2008.

[32] Francesco Marcelloni and Massimo Vecchio. An efficient lossless compression algorithm for tiny nodes of monitoring wireless sensor networks. *The Computer Journal*, 52(8):969–987, 2009.

[33] Markus F.X.J. Oberhumer . LempelZivOberhumer. http://www.oberhumer.com/opensource/lzo/.

[34] Nicholas Nethercote, Robert Walsh, and Jeremy Fitzhardinge. Building workload characterization tools with valgrind. In *IEEE Symposium on Workload Characterization*, page 2, 2006.

[35] Liadan O'callaghan, Nina Mishra, Adam Meyerson, Sudipto Guha, and Rajeev Motwani. Streaming-data algorithms for high-quality clustering. In *Proceedings 18th International Conference on Data Engineering*, pages 685–694. IEEE, 2002.

[36] A Oprea, J Courbat, N Bârsan, D Briand, NF De Rooij, and U Weimar. Temperature, humidity and gas sensors integrated on plastic foil for low power applications. *Sensors and Actuators B: Chemical*, 140(1):227–232, 2009.

[37] Ngoc Duy Pham, Trong Duc Le, and Hyunseung Choo. Enhance exploring temporal correlation for data collection in wsns. In *2008 IEEE International Conference on Research, Innovation and Vision for the Future in Computing and Communication Technologies*, pages 204–208. IEEE, 2008.

[38] George M Phillips. *Interpolation and approximation by polynomials*, volume 14. Springer Science & Business Media, 2003.

[39] James Pope, Antonis Vafeas, Atis Elsts, George Oikonomou, Robert Piechocki, and Ian Craddock. An accelerometer lossless compression algorithm and energy analysis for iot devices. In *2018 IEEE Wireless Communications and Networking Conference Workshops (WCNCW)*, pages 396–401. IEEE, 2018.

[40] Christopher M Sadler and Margaret Martonosi. Data compression algorithms for energy-constrained devices in delay tolerant networks. In *Proceedings of the 4th*

international conference on Embedded networked sensor systems, pages 265–278. ACM, 2006.

[41] Omid Sarbishei. On the accuracy improvement of low-power orientation filters using IMU and MARG sensor arrays. In *IEEE International Symposium on Circuits and Systems*, pages 1542–1545, 2016.

[42] Khalid Sayood. *Introduction to data compression*. Morgan Kaufmann, 2017.

[43] Thomas Schoellhammer, Eric Osterweil, Ben Greenstein, Mike Wimbrow, and Deborah Estrin. Lightweight temporal compression of microclimate datasets [wireless sensor networks]. *29th Annual IEEE International Conference on Local Computer Networks*, pages 516–524, 2004.

[44] Michael Ian Shamos and Dan Hoey. Geometric intersection problems. In *17th Annual Symposium on Foundations of Computer Science (sfcs 1976)*, pages 208–215. IEEE, 1976.

[45] Senthil Shanmugasundaram and Robert Lourdusamy. A comparative study of text compression algorithms. *International Journal of Wisdom Based Computing*, 1(3):68–76, 2011.

[46] Heung-Yeung Shum, Sing Bing Kang, and Shing-Chow Chan. Survey of image-based representations and compression techniques. *IEEE transactions on circuits and systems for video technology*, 13(11):1020–1037, 2003.

[47] Souvaine, Diane. Line segment intersection using a sweep line algorithm. http://www.cs.tufts.edu/comp/163/notes05/seg_intersection_handout.pdf. Computational Geometry, Tufts University, 2008.

[48] James A Storer and Thomas G Szymanski. Data compression via textural substitution. *Journal of the ACM*, 29(4):928–951, 1982.

[49] Jukka Teuhola. A compression method for clustered bit-vectors. *Information processing letters*, 7(6):308–311, 1978.

[50] Jeffrey S Vitter. Random sampling with a reservoir. *ACM Transactions on Mathematical Software (TOMS)*, 11(1):37–57, 1985.

[51] Jeffrey Scott Vitter. Faster methods for random sampling. *Communications of the ACM*, 27(7):703–718, 1984.

[52] Getinet Woyessa, Kristian Nielsen, Alessio Stefani, Christos Markos, and Ole Bang. Temperature insensitive hysteresis free highly sensitive polymer optical fiber bragg grating humidity sensor. *Optics express*, 24(2):1206–1213, 2016.

[53] Hanaa ZainEldin, Mostafa A Elhosseini, and Hesham A Ali. Image compression algorithms in wireless multimedia sensor networks: A survey. *Ain Shams engineering journal*, 6(2):481–490, 2015.

[54] Jacob Ziv and Abraham Lempel. A universal algorithm for sequential data compression. *IEEE Transactions on information theory*, 23(3):337–343, 1977.

[55] Jacob Ziv and Abraham Lempel. Compression of individual sequences via variable-rate coding. *IEEE transactions on Information Theory*, 24(5):530–536, 1978.

[56] Davide Zordan, Borja Martinez, Ignasi Vilajosana, and Michele Rossi. To compress or not to compress: Processing vs transmission tradeoffs for energy constrained sensor networking. *arXiv preprint arXiv:1206.2129*, 2012.

[57] Davide Zordan, Borja Martinez, Ignasi Vilajosana, and Michele Rossi. On the performance of lossy compression schemes for energy constrained sensor networking. *ACM Transactions on Sensor Networks (TOSN)*, 11(1):15, 2014.