Sampling, qualification and analysis of data streams

Rayane El Sibai

THÈSE DE DOCTORAT

Préparée au

Laboratoire LISITE - Institut supérieur d'électronique de Paris (ISEP)

École doctorale Informatique, télécommunications et électronique (Paris)

par

Rayane EL SIBAI

pour l'obtention du grade de

Docteur de l'Université Pierre et Marie Curie (Paris) - Sorbonne Université

Spécialité : **Informatique**

présentée et soutenue publiquement le **4 Juillet 2018**

Sampling, qualification and analysis of data streams

Directeurs de thèse	
Raja CHIKY	HDR, ISEP
Kablan BARBAR	HDR, Université Libanaise
Encadrants de thèse	
Yousra CHABCHOUB	MCF, ISEP
Jacques DEMERJIAN	MCF, Université Libanaise
Rapporteurs	
Mustapha LEBBAH	HDR, Université Paris 13
Vincent LEMAIRE	HDR, Orange Labs
Examinateurs	
Bernd AMANN	HDR, UPMC - Sorbonne Université
Karine ZEITOUNI	HDR, University of Versailles St Quentin (UVSQ)

"If you can dream it, you can do it."

Walt Disney

"Là où la volonté est grande, les difficultés diminuent. "

Nicolas Machiavel

Remerciements

"Soyons reconnaissants aux personnes qui nous donnent du bonheur ; elles sont les charmants jardiniers par qui nos âmes sont fleuries." [Marcel Proust]

Je voudrais tout d'abord adresser ma reconnaissance perpétuelle à mon co-encadrant de thèse M. Jacques Demerjian, qui m'a donné l'opportunité de faire cette thèse. Il a toujours été disponible pour me soutenir, me conseiller et répondre à mes questions tout au long de cette thèse.

Les mots ne suffisent pas à remercier ma co-encadrante de thèse Mme. Yousra Chabchoub, à qui je dois exprimer ma gratitude éternelle pour ses conseils précieux, nos discussions fructueuses et sa sympathie. Elle a toujours trouvé le temps pour m'encadrer et me guider durant la thèse.

Je remercie profondément mes directeurs de thèse Mme. Raja Chiky et M. Kablan Barbar pour la confiance qu'ils m'ont accordée en acceptant de diriger ma thèse.

Je tiens à remercier sincèrement M. Vincent Lemaire et M. Mustapha Lebbah d'avoir accepté d'être rapporteurs de ce manuscrit. Je remercie également Mme. Karine Zeitouni, M. Bernd Amann et M. Vincent Guigue d'avoir accepté de participer au jury de cette thèse.

Je tiens à saisir cette occasion et remercier M. Eric Gressier Soudan pour les suggestions et conseils qu'il m'a donnés durant la soutenance de mi-parcours.

Ce fût un grand plaisir de collaborer avec Mme. Zakia Kazi-Aoul, Mme. Christine Fricker et M. Jacques Bou Abdo, que je remercie vivement pour leurs temps et leurs commentaires utiles.

Mes plus chaleureux remerciements s'adressent à mes collègues à l'SEP, en particulier, Denis, Loic, Manuel, Sathiya, et Xiang Nan. Merci pour la géniale ambiance et la convivialité. Un merci tout particulier à Amadou avec qui j'ai partagé ce voyage de doctorat.

Finalement, un grand merci à mes très chers parents pour leur confiance, leur affection et leurs sacrifices. Cette expérience n'aurait pas été possible sans eux.

Cette thèse est dédiée à mes parents
pour leur soutien, leur encouragement et leur amour
inconditionnel.

Résumé

Un système de surveillance environnementale collecte et analyse continuellement les flux de données générés par les capteurs environnementaux. L'objectif du processus de surveillance est de filtrer les informations utiles et fiables et d'inférer de nouvelles connaissances qui aident l'exploitant à prendre rapidement les bonnes décisions. L'ensemble de ce processus, de la collecte à l'analyse des données, soulève deux problèmes majeurs : le volume de données et la qualité des données.

D'une part, le débit des flux de données générés n'a pas cessé d'augmenter sur les dernières années, engendrant un volume important de données continuellement envoyées au système de surveillance. Le taux d'arrivée des données est très élevé par rapport aux capacités de traitement et de stockage disponibles du système de surveillance. Ainsi, un stockage permanent et exhaustif des données est très coûteux, voire parfois impossible. D'autre part, dans un monde réel tel que les environnements des capteurs, les données sont souvent de mauvaise qualité, elles contiennent des valeurs bruitées, erronées et manquantes, ce qui peut conduire à des résultats défectueux et erronés.

Dans cette thèse, nous proposons une solution appelée filtrage natif, pour traiter les problèmes de qualité et de volume de données. Dès la réception des données des flux, la qualité des données sera évaluée et améliorée en temps réel en se basant sur un modèle de gestion de la qualité des données que nous proposons également dans cette thèse. Une fois qualifiées, les données seront résumées en utilisant des algorithmes d'échantillonnage. En particulier, nous nous sommes intéressés à l'analyse de l'algorithme Chain-sample que nous comparons à d'autres algorithmes de référence comme l'échantillonnage probabiliste, l'échantillonnage déterministe et l'échantillonnage pondéré. Nous proposons aussi deux nouvelles versions de l'algorithme Chain-sample améliorant sensiblement son temps d'exécution.

L'analyse des données du flux est également abordée dans cette thèse. Nous nous intéressons particulièrement à la détection des anomalies. Deux algorithmes sont étudiés : Moran scatterplot pour la détection des anomalies spatiales et CUSUM pour la détection des anomalies temporelles. Nous avons conçu une méthode améliorant l'estimation de l'instant de début et de fin de l'anomalie détectée dans CUSUM.

Nos travaux ont été validés par des simulations et aussi par des expérimentations sur deux jeux de données réels et différents : Les données issues des capteurs dans le réseau de distribution de l'eau potable fournies dans le cadre du projet Waves et les données relatives au système de vélo en libre-service (Velib).

Mots-clés : Flux de données, Algorithmes d'échantillonnage, Qualité des données, Analyse des données, Cloud computing.

Abstract

An environmental monitoring system continuously collects and analyzes the data streams generated by environmental sensors. The goal of the monitoring process is to filter out useful and reliable information and to infer new knowledge that helps the network operator to make quickly the right decisions. This whole process, from the data collection to the data analysis, will lead to two keys problems: data volume and data quality.

On the one hand, the throughput of the data streams generated has not stopped increasing over the last years, generating a large volume of data continuously sent to the monitoring system. The data arrival rate is very high compared to the available processing and storage capacities of the monitoring system. Thus, permanent and exhaustive storage of data is very expensive, sometimes impossible. On the other hand, in a real world such as sensor environments, the data are often dirty, they contain noisy, erroneous and missing values, which can lead to faulty and defective results.

In this thesis, we propose a solution called native filtering, to deal with the problems of quality and data volume. Upon receipt of the data streams, the quality of the data will be evaluated and improved in real-time based on a data quality management model that we also propose in this thesis. Once qualified, the data will be summarized using sampling algorithms. In particular, we focus on the analysis of the Chain-sample algorithm that we compare against other reference algorithms such as probabilistic sampling, deterministic sampling, and weighted sampling. We also propose two new versions of the Chain-sample algorithm that significantly improve its execution time.

Data streams analysis is also discussed in this thesis. We are particularly interested in anomaly detection. Two algorithms are studied: Moran scatterplot for the detection of spatial anomalies and CUSUM for the detection of temporal anomalies. We have designed a method that improves the estimation of the start time and end time of the anomaly detected in CUSUM.

Our work was validated by simulations and also by experimentation on two real and different data sets: The data issued from sensors in the water distribution

network provided as part of the Waves project and the data relative to the bike sharing system (Velib).

keywords: Data streams, Sampling algorithms, Data quality, Data analysis, Cloud computing.

Contents

Introduction 1
Context and motivation . . . 1
Contributions and organization of the manuscript . . . 4
Application domains . . . 6
List of publications . . . 8

I Data streams summarization 11

1 Data streams sampling algorithms 13
1.1 Introduction . . . 14
1.2 Data streams basic concepts . . . 14
1.2.1 Definition . . . 14
1.2.2 Data streams structure . . . 15
1.2.3 Time modeling and windowing models . . . 16
1.3 Data streams application domains . . . 18
1.3.1 Sensor networks . . . 18
1.3.2 Financial analysis . . . 19
1.3.3 Network traffic analysis . . . 19
1.4 Data streams management . . . 19
1.4.1 Data streams management system characteristics . . . 20
1.4.2 Data streams management systems . . . 20
1.5 Sampling algorithms . . . 22
1.6 Discussion . . . 28

2 Chain-sample algorithm for data streams sampling over a sliding window 31
2.1 Introduction . . . 32
2.2 The traditional Chain-sample algorithm . . . 32
2.2.1 Motivation . . . 32
2.2.2 Algorithm description . . . 33
2.2.3 Collision problem . . . 33

2.3 Chain+: A free redundancy Chain-sample algorithm 35
2.3.1 Construction of the samples 35
2.3.2 Memory usage for a single *chain-sample* 36
2.3.3 Trade-off between the execution time, sampling rate and window size 38
2.3.4 Comparison of the Chain+ sampling against the Simple Random Sampling algorithm 39
2.4 Enhancing the Chain+ sampling algorithm 42
2.4.1 Inverting the selection for high sampling rates strategy 42
2.4.2 Divide-to-Conquer strategy 43
2.4.3 Experimentations 45
2.5 Conclusion 46

3 On the impact of data sampling on data streams statistics inference 49
3.1 Introduction 50
3.2 Sampling impact on the queries estimation accuracy 50
3.2.1 Chain+ sampling algorithm 51
3.2.2 A bounded-space SRS algorithm without replacement over a purely sliding window 51
3.2.3 Deterministic sampling algorithm 52
3.2.4 Experimentations 53
3.3 Sampling impact on the anomalies detection 57
3.3.1 Problem definition 57
3.3.2 EWMA control chart algorithm 59
3.3.3 A bounded-space SRS algorithm without replacement over a jumping sliding window 59
3.3.4 A bounded-space WRS algorithm without replacement over a jumping sliding window 60
3.3.5 Experimentations 61
3.4 Conclusion 65

II Managing data quality in streaming sensor networks 67

4 Modeling data quality in sensors networks 69
4.1 Introduction 69
4.2 Data quality basic concepts 70
4.3 Data quality management in sensor networks 74
4.3.1 Data quality dimensions in sensor networks 74
4.3.2 Sensor data quality management, a new approach 79
4.4 Data quality in streaming sensor networks, related works 80
4.5 Conclusion 84

III Data streams anomalies detection 85

5 An in-depth analysis of CUSUM algorithm for change detection in time series 87
5.1 Introduction 88
5.2 Anomalies detection algorithms 88

5.3 An analysis of CUSUM algorithm . . . 91
5.3.1 Algorithm description . . . 91
5.3.2 Choice of the parameters . . . 92
5.3.3 Variability of the Run Length (RL) . . . 94
5.4 Enhancing the reactivity of CUSUM algorithm . . . 95
5.4.1 Detecting the anomaly start time . . . 96
5.4.2 Detecting the anomaly end time . . . 97
5.5 Detecting mean change . . . 97
5.5.1 Efficiency metrics . . . 97
5.5.2 Experimentations . . . 99
5.6 Application to stuck-at error: Detecting variation change . . . 101
5.7 Conclusion . . . 104

6 Spatial outliers detection with Moran Scatterplot 107
6.1 Introduction . . . 107
6.2 Motivation . . . 108
6.3 Dataset description and problem definition . . . 110
6.4 Spatial outliers detection with an improved Moran scatterplot . . . 111
6.4.1 Moran scatterplot . . . 111
6.4.2 Improvement of Moran scatterplot using Gower's coefficient . 113
6.5 Enhancing resources distribution in Velib system . . . 115
6.6 Conclusion . . . 120

IV Data streams native filtering 121

7 Implementation of the data streams native filters solution 123
7.1 Introduction . . . 123
7.2 Implementing data streams sampling and qualification modules in WAVES platform . . . 124
7.2.1 WAVES FUI project . . . 124
7.2.2 Native filters module . . . 126
7.3 Information technology infrastructure for data streams native filtering 129
7.3.1 Resources consumption in native filters . . . 129
7.3.2 Moving to the Cloud computing . . . 137
7.4 Conclusion . . . 142

Conclusion and Perspectives 145

References 149

List of Figures

0.1 Native filtering of data streams. 2
0.2 Links between the chapters of the thesis. 5
0.3 Volume of the water consumed by the sector, over time. 7

1.1 Windowing models of data streams. 17

2.1 Sampling $k = 1$ item over a sliding window of size $n = 4$, using the Chain-sample algorithm. 34
2.2 Collision rate. 35
2.3 Impact of the window size on the *chain-sample* length. 38
2.4 Variation of the *chain-sample* length over time. 38
2.5 Impact of the window size on the execution time of the Chain+ sampling algorithm, for different sampling rates k/n. 39
2.6 Impact of the sampling rate on the execution time of the Chain+ sampling algorithm, for different windows sizes n. 40
2.7 Execution time of the Simple Random Sampling and Chain+ sampling algorithms, for different sampling rates k/n. 41
2.8 Distribution of the sample size of the Simple Random Sampling algorithm without replacement. 42
2.9 Sampling $k = 6$ items over a sliding window of size $n = 10$ with the Chain+ sampling algorithm using the "Inverting the selection for high sampling rates" strategy. 43
2.10 Sampling $k = 5$ items over a sliding window of size $n = 10$ with the Chain+ sampling algorithm using the "Divide-to-Conquer" strategy. 45
2.11 Impact of the "Inverting the selection for high sampling rates" strategy on the execution time of the Chain+ sampling algorithm. 46

3.1 Execution time of the Deterministic, Simple Random and Chain+ sampling algorithms over a purely sliding window, for different sampling rates k/n. 54
3.2 Impact of the sampling rate and window size on the mean estimation error of the Chain+ sampling algorithm, for different sampling rates k/n 56

3.3 Impact of the collision problem on the mean estimation error of the Chain-sample algorithm, for different sampling rates k/n. 56
3.4 Mean estimation error of the Deterministic, Simple Random and Chain+ sampling algorithms for a sampling rate of 1 observation per 12 hours, for different window sizes. 57
3.5 Experiments' strategy. 58
3.6 Sum of the sampled flowmeters of a sector when sampling $k = 5$ items over a jumping sliding window of size $n = 10$ using WRS algorithm. . . . 62
3.7 Volume of the water consumed by the sector, over time, with and without anomalies. 63
3.8 EWMA control chart. 63
3.9 Impact of the sampling rate on the anomaly detection performance according to the sampling algorithm, for the first scenario. 64
3.10 Comparison of the sampling impact on the anomaly detection performance when using the Simple Random and Weighted Random Sampling algorithms for both scenarii $S1$ and $S2$. 64

4.1 *"Journal and conference proceedings from ISI Web of Knowledge searched by a query title and business economics domain using the key words information quality or data quality, data quality and metadata, and data management"* (From [Moges, 2014]). 71
4.2 TDQM methodology. 72
4.3 Data quality dimensions, empirical approach [Strong et al., 1997]. . . . 74
4.4 Types of abnormal data. 76
4.5 Examples of outliers faults in the raw humidity readings in the NIMS deployment [Kaiser et al., 2005]. 77
4.6 Stuck-at faults in the chlorophyll concentrations from two buoys in the NAMOS deployment at Lake Fulmor monitoring the marine environment [Dhariwal et al., 2006]. 78
4.7 Sensors data quality management approach based on TDQM methodology. 80

5.1 $ARL_0(k, h)$: Impact of k and the control limit h on ARL_0. 93
5.2 $ARL_1(k, h)$: Impact of k and the control limit h on ARL_1. 93
5.3 Impact of the control limit h on ARL_δ. 94
5.4 Impact of the shift δ on ARL, $h \in [3, 5]$. 94
5.5 Impact of small shifts δ on ARL, $h \in [3, 5]$. 95
5.6 Variation of C_t over time. 99
5.7 Variation of ST counter over time. 100
5.8 Variation of ET counter over time. 100
5.9 Injection of variation change errors. 103
5.10 Variation v_t of s_t after the injection of errors. 103
5.11 Variation of C_t over time. 103

6.1 Distribution of the number of neighbors. 112
6.2 Distribution stations capacity. 113
6.3 Improved Moran scatterplot based on occupancy data of Velib system on Thursday $10/31/2013\ 10:00\ am$. 115
6.4 Number of trips over time. 116
6.5 Number of problematic stations. 116
6.6 Number of problematic stations in the day. 117

6.7 Detected spatial outliers. 118
6.8 Average number of problematic stations in the day after the users collaboartion. 118
6.9 Mean duration of stations invalidity. 119
6.10 Mean cumulative duration of stations invalidity. 119

7.1 WAVES platform architecture (from FUI17 WAVES Annexe technique). 125
7.2 Native filters architecture. 126
7.3 Execution time of the sampling process according to the number of streams, using the local server. 131
7.4 Execution time of the sampling process according to the number of streams, using the Cloud server. 131
7.5 Execution time of the cleaning process according to the number of streams. 132
7.6 Number of servers needed to ensure high availability, using the Cloud server. 133
7.7 Relational model of the summary. 134
7.8 Representation of a row in an HBase table as a multidimensional map. 136

List of Tables

1.1 Structured data stream tuples generated by a water operator. 16
1.2 Windowing models of data streams. 17
1.3 Weaknesses of data streams sampling algorithms. 29

2.1 Execution time reduction of Chain+ sampling algorithm using the "Divide-to-Conquer" strategy, for a sampling rate $k/n = 0.5$, a splitting factor $c = k$, and for different window sizes n 47

4.1 Taxonomy of sensor data faults from a data-centric view [Ni et al., 2009]. 76

5.1 Simulated Run Lengths (RL), for different shift sizes. 95
5.2 Obtained results for mean change detection. 101
5.3 Performance metrics of CUSUM. 101
5.4 Obtained results for stuck-at errors detection. 104
5.5 Performance metrics of CUSUM. 104

6.1 Number of detected outliers stations with the improved Moran scatterplot. 115

7.1 Number of observations that can be treated in 15 minutes, using the Cloud server. 132
7.2 Storage requirements of data streams summaries according to the sampling algorithm. 134
7.3 Decision table based on the Cloud computing criteria. 143

Introduction

Context and motivation

An environmental monitoring process consists of a continuous collection, analysis and reporting of observations or measurements of environmental characteristics. Different environmental components can be described and qualified (soil, air, water...) using different types of sensors. These latter perform regular measures that are sent to a central system to be analyzed using specific diagnostic tools. The final objective is to discover and infer new knowledge about the environment, in order to help the administrator to make the good decisions. A main purpose of the monitoring system is to detect the anomalies, also called "events". Different data mining techniques are applied to the collected data in order to infer in real-time aggregated statistics useful for anomalies detection and forecasting purposes. This process helps the administrator to supervise the observed system and to take quickly the right decisions. The whole process, from the data collection to data analysis, leads to two major problems: the management of the data volume and the quality of this data.

On the one hand, a sensor generates the data in the form of a stream that consists of a large volume of data sent to the monitoring system continuously. The arrival rate of the data is very high compared to the available processing and storage capacities. The monitoring system is thus faced with a large amount of data for which permanent and exhaustive storage is very expensive and sometimes impossible. That's why we need to process the data stream in one pass, without storing it. However, for a particular stream, it is not always possible to predict in advance all the processing to be performed. On the other hand, in a real-world such as sensor environment, the data are often dirty, they contain noisy, erroneous, duplicate, and/or missing values. This is due to many factors: local interference, malicious nodes, network congestion, limited sensor accuracy, harsh environment, sensor failure or malfunction, calibration error, and insufficient sensor battery. As in any data analysis process, the conclusions and decisions based on these data may

be faulty or erroneous if the data are of poor quality.

Our goal in this thesis is to treat the chain, from the data collection to the anomalies (events) detection of data streams generated by sensors.

As a first step, we propose the native filtering of data streams as a solution to overcome the two problems related to the data collection: the huge volume of generated data and their poor quality. This solution consists of filtering the data qualitatively (evaluating and improving the quality of the received data), and then, quantitatively (summarizing the data), as shown in Figure 0.1.

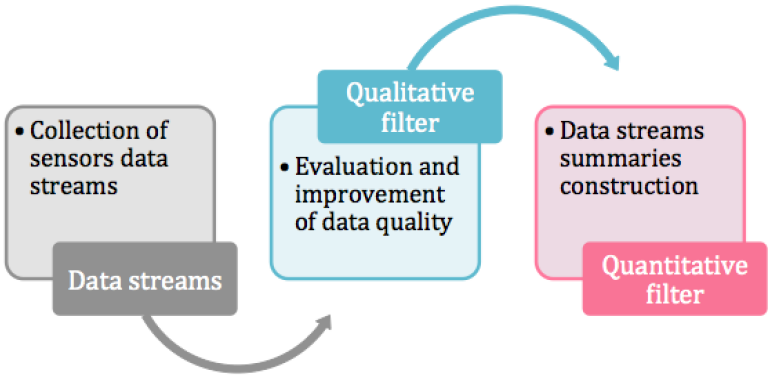


FIGURE 0.1 – Native filtering of data streams.

Qualitative filter

One solution to overcome the problem of poor quality of the data is to use sensors with high precision to neglect the potential errors that could occur. Another alternative is to deploy redundant sensors to cover sensor failure. However, these approaches are very expensive. In this thesis, we propose an approach that consists of using methods and algorithms to first evaluate the quality of the data and then to improve it in order to obtain reliable and effective results.

Several research studies have focused on the management of data quality in sensor networks. [Jeffery et al., 2006] introduced the so-called Extensible Sensor Stream Processing (ESP) system to clean sensors data. The system detects erroneous data, replaces missing data, and deletes duplicate data. [Lim et al., 2009] proposed to evaluate the accuracy of the data by calculating the difference between the experimental distribution and the theoretical distribution of the data. A data cleaning system based on machine learning algorithms has been proposed by [Ramirez et al., 2011]. It calculates the difference between the received data and the predicted data to evaluate the accuracy of the data. In [Klein et al., 2007], the authors proposed a model for evaluating and storing the information about the accuracy and completeness of the data. We present in Chapter 4 a state of the art of the existing approaches for the qualification of sensors data. Then, we introduce the model we propose to evaluate and to improve the quality of the data in the context of sensor networks.

Quantitative filter

Data streams are volatile, once expired, they are no longer available for analysis. This makes it impossible to evaluate any undefined query before the arrival of the

data, while new requirements may appear after the arrival of the stream. In this case, the data stream management system cannot answer the new queries. One solution to overcome this problem is to store an extract of the stream in a compact structure, called summary.

The challenge is to decide which data to store in order to maintain a representative summary of the entire stream. One of the structures used to preserve the stream history is the general summary [Midas et al., 2010]. It is a data structure updated whenever new data arrive. Its particularity lies in its ability to carry out analysis tasks on the data of the stream and to give an approximate answer to any query and for any investigated period of time. These characteristics distinguish the general summary from other data streams summaries called "synopsis" such as Sketches. The Sketch is a very compact data stream summary used to answer specific queries about data stream. We can mention the Count-Min sketch [Cormode and Muthukrishnan, 2005] used to estimate the frequency of an element, and the sketch of Flajolet-Martin [Flajolet and Martin, 1985] which estimates the number of distinct elements in a data stream.

The effectiveness of a general summary is measured in terms of the accuracy of the provided response, the memory space to store it, and the time to update it [Midas et al., 2010]. The challenge is to decide what to store in this summary and how to ensure that the summary can meet the requirements of the application while respecting the available resources of the system.

Sampling methods can be used to construct a general summary of data streams. Two categories of these techniques are provided in the literature: probabilistic methods and deterministic methods. Probabilistic methods also called stochastic methods are characterized by the fact that each element has a probability of inclusion in the sample. The composition of the obtained sample is thus random. Simple Random Sampling (SRS) and Stratified sampling are two examples of random sampling. For deterministic methods, there is no randomness in the composition of the sample: for example, selecting all the elements having even indexes. The choice of the appropriate sampling method depends, of course, on the application and the purpose of the sampling. We first present in Chapter 1 the state of the art of sampling methods. Then, we focus in Chapter 2 on the study of the Chain-sample, a probabilistic sampling algorithm well adapted to the context of data streams.

Anomalies detection

Native filtering (qualitative and quantitative), presented above, is a pre-processing step that prepares the data to be analyzed and exploited. In the data analysis phase, we are particularly interested in anomalies detection in data streams. This problem is addressed in several applications domains such as fraud detection for credit cards, intrusion detection in networks, image processing, etc. In sensor networks, anomalies detection can be used for many tasks such as fault diagnosis, intrusion detection, and applications monitoring [Chandola et al., 2009].

In sensor networks, there are two different types of anomalies: temporal and spatial. Indeed, sensor data has two characteristics: temporal and spatial correlation. Temporal correlation is due to the continuity of the observed measure. It implies that, for a single data stream, the data value at a given moment is often related

to the values measured at close moments. Spatial correlation consists in a strong relation between the values measured at the same time by nearby sensors. These two types of anomalies are studied in this thesis in Chapters 5 and 6 respectively.

Anomalies detection techniques in a temporal context can be classified into two categories: parametric and non-parametric methods. Parametric methods assume that the data follow a known probability distribution. Anomalies are defined as data having a low probability to belong to this distribution. On the contrary, non-parametric techniques do not make any assumption about data distribution. In this kind of methods, no a priori knowledge about data is needed. In sensor networks, non-parametric methods are frequently used. Indeed, in such environments, the data distribution can often change due to sensor resource constraints. The main non-parametric approaches used to detect anomalies in sensor networks are rule-based approaches, control chart methods (i.e. CUmulative SUM (CUSUM), Exponential Weighted Moving Average (EWMA)), clustering and support vector machine approaches. In this thesis, we study in detail the CUSUM algorithm in Chapter 5.

Several algorithms have been developed to detect anomalies in a spatial context. Among these algorithms, we mention the quantitative algorithms and the graphical algorithms. Quantitative methods perform statistical tests to distinguish the anomalies from the rest of the data, while the graphical algorithms are based on visualization. They present for each spatial point the distribution of its neighbors and identify the anomalies as isolated points, in specific regions. We are interested in Chapter 5 in Moran scatterplot, a data visualization method that exploits the spatial correlation.

Contributions and organization of the manuscript

As shown in Figure 0.2, this thesis deals with two main issues: native filtering and data stream analysis, and consists of four parts. We discuss the quantitative and qualitative filtering of data streams in the first two parts entitled "Data streams summarization" and "Managing data quality in streaming sensors networks" respectively. The implementation of the native filters solution is presented in the fourth part entitled "Data streams native filtering". The detection of anomalies in data streams is discussed in the third part entitled "Data streams anomalies detection".

We present in Chapter 1 the state of the art of different sampling algorithms used in data streams environments. We propose to classify these algorithms according to the following metrics: the number of passes over the data, the memory consumption, the skewing ability, and the resources consumption of the algorithm.

In Chapter 2, we study in detail the Chain-sample algorithm. We identify a particular weakness of this algorithm caused by the problem of collisions and redundancy of the items in the sample when the sampling rate is high. In order to overcome this problem, we modify the Chain-sample algorithm to improve the quality of the sample by eliminating the redundancy. We also propose two techniques to significantly reduce the execution time of the algorithm, even for a high sampling rate.

We address in Chapter 3 the impact of data sampling on events detection. Several

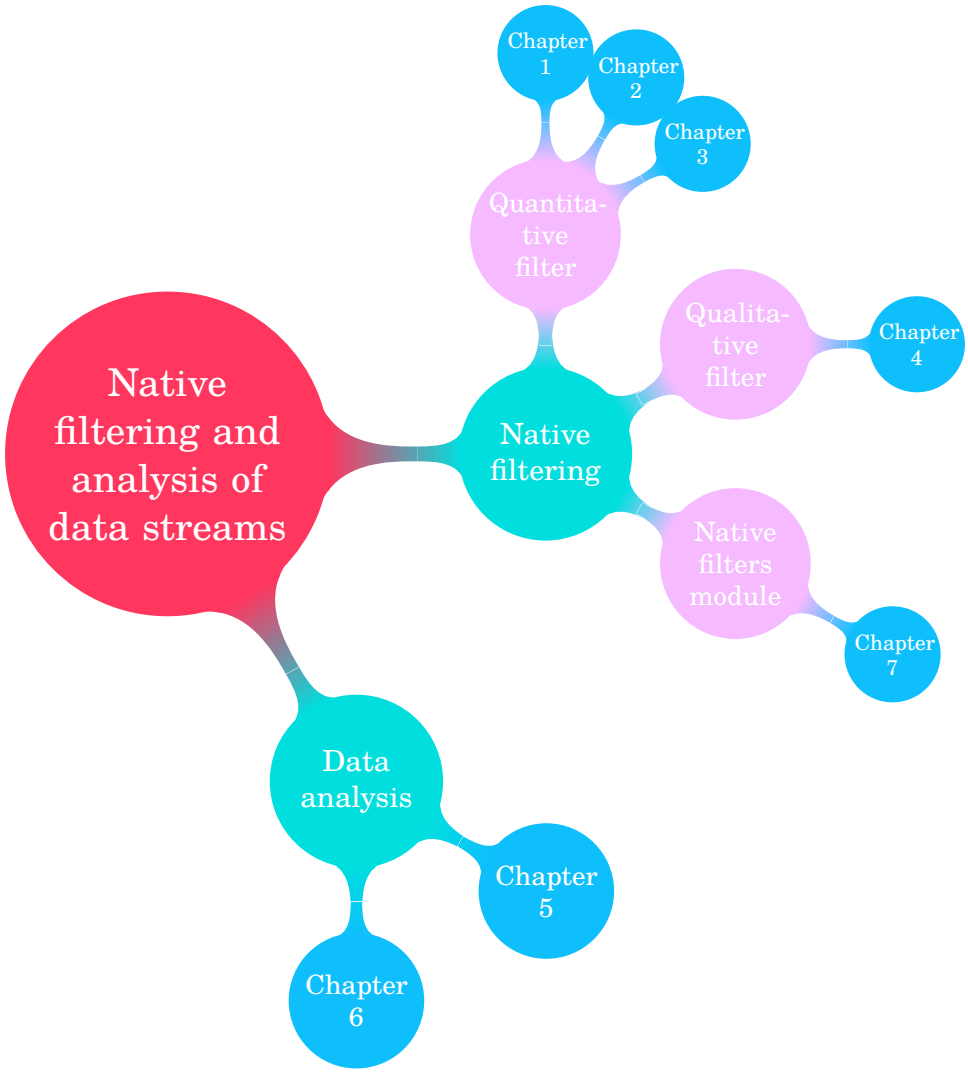


FIGURE 0.2 – Links between the chapters of the thesis.

sampling algorithms are performed on data streams: Deterministic sampling, Chain-sample, Simple Random Sampling (SRS) and Weighted Random Sampling (WRS). First, we adapt the SRS algorithm to the stream context by adapting it to the sliding window model. Then, we compare the performance of these algorithms in terms of execution time and accuracy of the queries answers. Thereafter, we study the impact of the sampling process on anomalies detection. In this context, the comparison of the algorithms is based on their response time in case of anomaly and the relevance of the detected anomalies.

We discuss in Chapter 4 the data quality aspects in dynamic environments, especially, in sensor networks. At first, we present the general definitions of data quality dimensions. Then, we detail the different dimensions of sensor data quality, and we provide our definitions for the $accuracy$ and $confidence$ dimensions. We also propose a new model for managing data quality in sensor networks. Compared to existing approaches, our model takes into account the errors caused by sensor faults.

In Chapter 5, we study in depth the CUSUM algorithm which allows detecting the temporal anomalies in a data stream issued from a single source. In particular, we analyze the choice of the parameters of the algorithm in order to achieve two

objectives. (1) Minimize the false positives: when the process is under control, the CUSUM algorithm should not detect any change, (2) detect quickly any deviation of the process. We also propose a method which determines with a good precision, the start and end time of the deviation of the process parameters.

In Chapter 6, we are interested in spatial anomalies detection methods, especially, in Moran scatterplot, a graphical algorithm based on visualization that exploits the similarity between spatial neighbors in order to detect spatial anomalies. At first, we propose an improved version of Moran scatterplot, in which, we enhance the definition of the weight matrix involved in the calculation of the distance between an observed value and its neighboring observations. We propose to calculate the weights based on several parameters related to the spatial points characteristics which qualify the correlation between them.

We present in Chapter 7 the native filters module which provides two features: real-time data streams qualification and sampling. Upon receipt of data streams from multiple sensors, the qualitative filter evaluates and improves the data quality based on the architecture presented in Chapter 4. Once the data are qualified, the quantitative filter proceeds to summarize the data using sampling algorithms. Several simultaneous data streams can be processed by the native filters module which in turn adapts to the characteristics of each stream. The integration of the native filters module into the WAVES project platform is also discussed in this chapter. We also evaluate in Chapter 7 the native filters solution in terms of the required computing resources. We present a benchmark of the Information Technology (IT) resources requirements while examining two network architectures for data processing: local computing and Cloud computing, and two infrastructures for data streams summaries storage: database and Hadoop. The considered computing resources are the execution time of the data streams qualification and sampling processes, and the memory storage required for storing the data streams summaries. We finally discuss in this chapter the migration benefit of the native filters solution to the Cloud computing environment.

Application domains

All the propositions presented in this thesis are tested and validated against real datasets issued from these application domains:

A. WAVES dataset

This thesis is part of the FUI 17 WAVES project, which aims to design and to develop a monitoring platform for the supervision of water distribution networks. The increase of the water stress in many parts of the world and the awareness of the value of fresh water as a scarce source require a reduction in the water losses along the production chain, from the natural water resources to the consumers. According to the Cador report [1] on the state of the heritage in France, the losses and leakage represent 30% of the total volume of water flowing into the water distribution

[1] http://www.economie.eaufrance.fr/IMG/pdf/Patrimoine_des_canalisations_d_AEP_France.pdf

network.

In order to supervise and to manage the water distribution network, many flowmeters have been deployed by the water operators. They periodically measure and send to the central monitoring system the instantaneous values of different water-related observables such as flow, pressure, and chlore. A large geographical area is divided into several sectors with several flowmeters deployed on the periphery of each sector. All these information are aggregated and analyzed by the monitoring system, which infers, in particular, the water consumption of the considered sector. The water consumption of each sector is indeed a key parameter for the detection of leaks. The volume of the water consumed by a given sector is calculated in real-time as an algebraic sum of the flows sent by its associated flowmeters. Each of these deployed flowmeters has two categories of attributes: spatial attributes and non-spatial attributes. The spatial attributes depict the geographical location of the flowmeter: latitude and longitude, while the non-spatial attributes include the name, ID, and the record observations of the flowmeter, and the diameter of the flowmeter.

The first dataset we are going to explore in this thesis is issued from the deployed flowmeters. The flow measurements for a given flowmeter are very variable. Indeed, they depend on the other flowmeters supplying the associated sector. For example, a given source associated with a particular flowmeter may suddenly stop supplying water to a sector. The latter will be delivered by its other associated peripheral sources.

The data recorded by the flowmeters are structured data streams and have both spatial and temporal characteristics. Each record observation is composed of two fields: the timestamp designating the recording date of the measure, and the value of the measure. These data are regularly generated by the sensors with a frequency of one observation every 15 minutes. Figure 0.3 shows the volume of the consumed water of a specific sector during five working days in January 2014. It is inferred in real-time as an algebraic sum of the flows delivered by its associated flowmeters, in m^3. One can notice a periodicity in the water consumption, related to the human activity. As it is a working day, we can notice two main peaks of consumption: an important peak in the morning and a second less important around $7\ pm$.

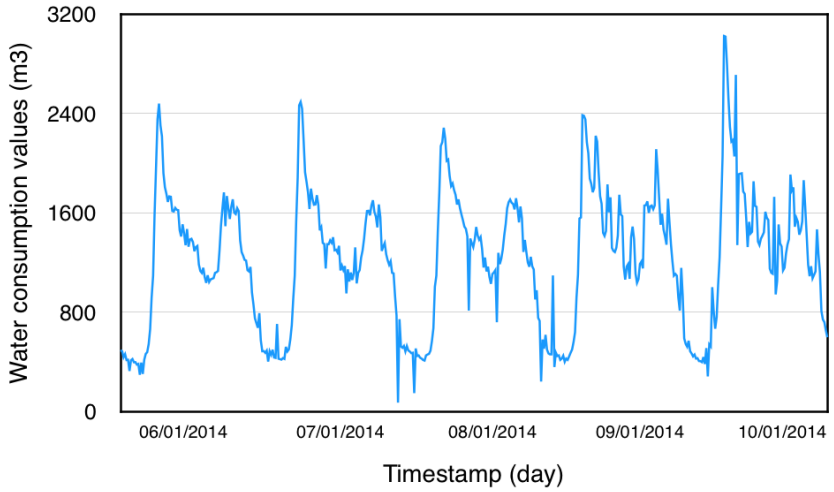


FIGURE 0.3 – Volume of the water consumed by the sector, over time.

B. Velib dataset

The second dataset we explored in this thesis is relative to the Parisian bike sharing system (Velib). These data are of two types: static data and dynamic data. The static data describe the Velib stations and they consist of two categories of attributes: spatial attributes and non-spatial attributes. The spatial attributes depict the geographical location of the station: latitude and longitude, while the non-spatial attributes include the ID of the station and its capacity (total number of docks). The dynamic data are of two kinds: occupancy data and trip data. Occupancy data are provided in real time. They represent the states of the stations in terms of the number of bikes present in each station for each timestamp t. This parameter is varying during the day and is closely dependent on users activity. Trip data depict the data corresponding to the trips of Velib' users. A trip is characterized by a departure and arrival timestamp, and a departure and arrival station. Trip data can be divided into two main categories: the working days and the weekends. Indeed, two days of the same category are very similar. In this thesis, we focus on the working days and we choose to analyze 24 hours trips: trips that took place on Thursday, October, the 31^{th}, 2013. This duration includes 121.709 trips, involving 1226 Velib stations.

List of publications

This thesis resulted in 8 publications in international conferences. In the following, our list of publications.

1. **Rayane El Sibai**, Yousra Chabchoub and Christine Fricker. *"Using spatial outliers detection to assess balancing mechanisms in bike sharing systems"*. In Proceedings of the 32^{th} IEEE International Conference on Advanced Information Networking and Applications (AINA), IEEE, May 2018.

2. **Rayane El Sibai**, Yousra Chabchoub, Raja Chiky, Jacques Demerjian and Kablan Barbar. *"An in-depth analysis of CUSUM algorithm for the detection of mean and variability deviation in time series"*. In Proceedings of the 16^{th} International Symposium on Web and Wireless Geographical Information Systems (W2GIS), Springer, May 2018.

3. **Rayane El Sibai**, Yousra Chabchoub, Raja Chiky, Jacques Demerjian and Kablan Barbar. *"Information Technology Infrastructure for Data Streams Native Filtering"*. In Proceedings of the IEEE Middle East & North Africa COMMunications Conference, IEEE, April 2018.

4. **Rayane El Sibai**, Yousra Chabchoub, Raja Chiky, Jacques Demerjian and Kablan Barbar. *"A performance evaluation of data streams sampling algorithms over a sliding window"*. In Proceedings of the IEEE Middle East & North Africa COMMunications Conference, IEEE, April 2018.

5. **Rayane El Sibai**, Yousra Chabchoub, Raja Chiky, Jacques Demerjian and Kablan Barbar. *"Assessing and Improving Sensors Data Quality in Streaming*

Context". In Proceedings of the 9^{th} International Conference on Computational Collective Intelligence Technologies and Applications (ICCCI), pages 590-599, Springer, September 2017.

6. **Rayane El Sibai**, Yousra Chabchoub, Jacques Demerjian, Zakia Kazi-Aoul and Kablan Barbar. *"Sampling algorithms in data stream environments"*. In the International Conference on Digital Economy (ICDEc), pages 29-36, IEEE, April 2016.

7. **Rayane El Sibai**, Yousra Chabchoub, Jacques Demerjian, Zakia Kazi-Aoul and Kablan Barbar. *"A performance study of the Chain sampling algorithm"*. In Proceedings of the 7^{th} International Conference on Intelligent Computing and Information Systems (ICICIS), pages 487-494, IEEE, December 2015.

8. Yousra Chabchoub, Zakia Kazi-Aoul, Amadou Fall Dia and **Rayane El Sibai**. *"On the dependencies of queries execution time and memory consumption in C-SPARQL"*. In Proceedings of the 12^{th} IADIS International Conference on Applied Computing (AC), pages 29-36, October 2015.

Part I

Data streams summarization

Chapter

1 Data streams sampling algorithms

Contents

1.1 Introduction **14**

1.2 Data streams basic concepts **14**

1.2.1 Definition 14

1.2.2 Data streams structure 15

1.2.3 Time modeling and windowing models 16

1.3 Data streams application domains **18**

1.3.1 Sensor networks 18

1.3.2 Financial analysis 19

1.3.3 Network traffic analysis 19

1.4 Data streams management **19**

1.4.1 Data streams management system characteristics 20

1.4.2 Data streams management systems 20

1.5 Sampling algorithms **22**

1.6 Discussion **28**

The scientific contribution presented in this chapter has been published in our paper: **Rayane El Sibai**, Yousra Chabchoub, Jacques Demerjian, Zakia Kazi-Aoul and Kablan Barbar. *"Sampling algorithms in data stream environments"*. In the International Conference on Digital Economy (ICDEc), pages 29-36, IEEE, April 2016.

1.1 Introduction

Data streams are large sets of data generated continuously and at a rapid rate in comparison to the available processing and storage capacities of the system that receives them. Thus, these streams cannot be fully stored. That is why we have to process them in one pass without storing them exhaustively. However, for a particular stream, it is not always possible to predict in advance all the processing to be performed. It is, therefore, necessary to save some of these data for future treatments. These stored data constitute the "summaries". Several techniques can be used for the construction of data streams summaries, among them the sampling algorithms.

In this chapter, we present a study of these algorithms. Firstly, we introduce the basic concepts of data streams, windowing models, as well as the data streams applications. Next, we detail the different sampling algorithms used in streaming environments, and we propose to qualify them according to the following metrics: the number of passes, memory consumption, skewing ability and complexity.

This chapter is organized as follows. We present in Section 1.2 the basic concepts of data streams. We discuss several applications domains of data streams in Section 1.3. Section 1.4 is dedicated to data streams management systems. In Section 1.5, we present a detailed study of the sampling algorithms used in streaming environments. We end the chapter with a discussion.

1.2 Data streams basic concepts

1.2.1 Definition

A data stream is an infinite sequence of tuples generated continuously and rapidly with respect to the available processing and storage capacities. Golab *et al.* [Golab and Özsu, 2003b] define a data stream as follows:

"*A data stream is a real-time, continuous, ordered (implicitly by arrival time or explicitly by timestamp) sequence of items. It is impossible to control the order in which items arrive, nor is it feasible to locally store a stream in its entirety.*"

Several other definitions of data streams have been presented in the literature. Their common characteristic is that they all rely on the main features of these streams, namely [Gabsi, 2011]:

- **Continuous.** The tuples arrive continuously and sequentially.
- **Fast.** The data arrive at a high speed compared to the processing and storage capacities available in the system that receives them.
- **Ordered.** The order of the data is often defined by timestamp which can be either implicit (data arrival time), or explicit by a timestamp contained in the data.

- **Unlimited volume.** The size of the data stream is potentially unbounded and can be very large. The exhaustive storage of all the received data is not possible. For instance, one gigabyte of records per hour is generated by AT&T (the largest provider of local and long distance voice and xDSL services in the United States) [Chakravarthy and Jiang, 2009].
- **Push-type.** The sources of the data are not controllable. They are programmed to send regular measurements. The input rate can vary widely from a data stream to another. Also, some data streams have irregular input rates, while others are highly bursty such as the HTTP traffic streams [Crovella and Bestavros, 1997] and the local Ethernet traffic streams [Leland et al., 1993].
- **Volatile.** Once the data are processed, they will be discarded, and there is no possibility to treat them another time unless they have been stored in memory. The latter is very small compared to the data stream size. Therefore, an immediate treatment of the data is required and has to be fast enough to achieve the response time requirement.
- **Uncertainty of the data.** Some data of the stream may be missing, duplicated, or prone to errors. This is due to external factors such as network congestion, hardware problem in measurement instrument, etc. (cf. Chapter 4).

1.2.2 Data streams structure

The form and type of the data belonging to a stream depend on the application that led to the creation of the data. Two types of data can be distinguished: quantitative data and qualitative data. The quantitative data includes the data whose representation is in the numerical form. They usually come from measurements. The qualitative data concern the data represented by specific values from a discrete set of possible values. For example, the weight of the human is a qualitative data that can be represented by the labels low, medium, high. Notice that the binary data are also considered as two-mode qualitative data ($0/1$, ON/OFF). Depending on the form of the data, the data stream can be represented by three types. In structured data streams, the tuples arrive as records that respect a specific relationship schema including the fields names of the tuples and the associated values. These tuples arrive in an ordered manner which is often determined by the timestamp of the tuple. An example of structured data stream tuples generated by a flowmeter is given in the Table 1.1. The data of a semi-structured stream are heterogeneous sets of weakly structured data, they arrive in the form of XML tags or RDF. An RDF data stream is an ordered sequence of pairs where each pair consists of a triple RDF and a timestamp. In unstructured data streams, the data have different structures. Currently, more than 85% of all business information are unstructured data [Blumberg and Atre, 2003]. These data include e-mails, surveys, Web pages, PowerPoint presentations, chats, etc. WebCQ [Liu et al., 2000] is a data stream management system for managing unstructured data streams. Its purpose is to monitor the pages on the Web in order to detect and to report the interesting changes that occur to the users.

Given that the types of data that can be handled by the data mining algorithms are restrained, a data pre-processing step is often necessary before proceeding to

TABLE 1.1 – Structured data stream tuples generated by a water operator.

Timestamp	**Sensor ID**	**Consumption** m^3
...	...	...
2014/12/31 00:00	Q 400G	196,19
2014/12/31 01:00	Q 400G	187,91
2014/12/31 02:00	Q 400G	188,24
2014/12/31 03:00	Q 400G	188,60
...	...	...

the data analysis phase. Data pre-processing aims at converting the raw data to a standard format adapted to the data mining algorithms. This step is also an opportunity to clean up the data by replacing missing data and regenerating the aberrant data. Finally, the standardization of the data is also necessary to bring all the data to the same definition domain to be able to compare their values independently of their original units.

1.2.3 Time modeling and windowing models

Data streams are infinite. They must be processed in an online manner, and the Data Stream Management System (DSMS) must provide fast responses to the continuous requests while respecting the data stream arrival rate. Thus, the windowing models were introduced in the formulation of the continuous requests in the DSMS. The data windowing models are based on the principle of cutting the stream into successive portions, and they are used to limit the amount of data to be processed. With the use of the windows, at any time, a finite set of tuples of the stream can be defined and used to respond to the query and produce the corresponding results. The windowing models can be classified according to the time modeling fashion. The temporal aspect of a data stream can be modeled in two manners: the physical time, also called temporal time, and the logical time, also called sequential time. The physical time is expressed in terms of date while the logical time is expressed in terms of the number of elements. One can notice that, with the logical time model, it is possible to know in advance the number of elements in the window. This number is unknown for the physical window model when the stream rate is variable. Alternatively, each of these two types of windows can be defined by its two boundaries. According to the start and end dates of the window, we can distinguish:

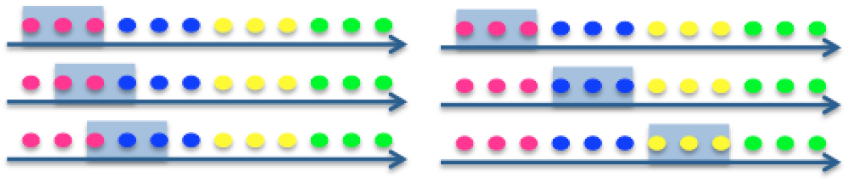
- **Fixed window.** When using the fixed window model, the stream is partitioned into non-overlapping windows and the data are preserved only for that part of the stream within the current window. The boundaries of this type of window are accurate and absolute.
- **Sliding window.** With the sliding window model, the boundaries of the window change over time, each time an item is added to the window, the oldest element will come out. The queries are periodically performed on the data included in the last window. There are three variants of the sliding window: the purely sliding window with which the offset between the successive windows is less

than the window size, the jumping window where the offset between successive windows is equal to the window size and the hopping window with an offset greater than the window size. A sliding window can be tuple-based (the most recent n elements) or time-based (elements received within the last δ minutes).

- **Landmark window.** The start date of this window is fixed, while the end date is relative. The size of the window increases gradually as new elements of the stream arrive. For instance, a window between a specific date and the actual date is of type landmark.

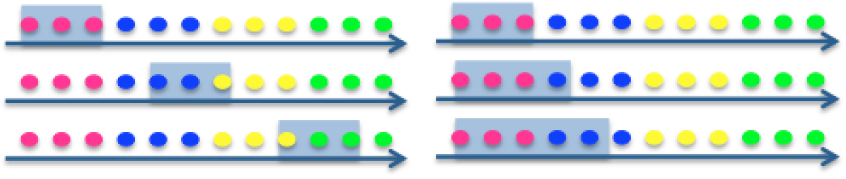
TABLE 1.2 – Windowing models of data streams.

Windowing model	Dates	Example
Physical fixed	Sequential	From the 19^{th} element to the 56^{th} element
Logical fixed	Temporal	From $01/01/2018$ to $30/01/2018$
Physical sliding	Sequential	The last 10 elements
Logical sliding	Temporal	The last 10 days
Physical landmark	Sequential	From the 30^{th} element to the last received element
Logical landmark	Temporal	From $12/03/2018$ to present



(a) Purely sliding window of size $n = 3$

(b) Jumping sliding window of size $n = 3$



(c) Hopping sliding window of size $n = 3$ and offset $o = 4$

(d) Landmark sliding window of initial size $n = 3$

FIGURE 1.1 – Windowing models of data streams.

1.3 Data streams application domains

The field of data streams is a subject of growing interest in the industrial community. This interest is reflected by the growing number of applications and industrial systems that continuously generate data streams [Golab and Özsu, 2003a, Golab and Özsu, 2003b]. These applications are heterogeneous and quite diverse, although their main purpose is the supervision and the control of the data. A typical application of data streams is to study the impact of the weather on the traffic networks. Such a study is useful for analyzing and predicting the traffic density as a function of the weather conditions [Gietl and Klemm, 2009]. The analysis of weather data is also used to predict the weather conditions. Indeed, several weather indicators, such as the temperature, humidity, air pressure, wind speed, etc, are indicative of the weather. Through the classification and learning of these data, several models can be derived and used to predict the future weather conditions [Bartok et al., 2010]. Social networks also provide more and more data streams that can be exploited in many areas. For instance, TwitterMonitor is a real-time system that allows the detection and the analysis of emerging topics on Twitter. The results are provided to the users who in turn interact with the system to rank the detected trends according to different criteria [Mathioudakis and Koudas, 2010]. Data streams can be found in other applications as well, such as website logs [Gilbert et al., 2001] and medical remote monitoring [Sachpazidis, 2002, Brettlecker and Schuldt, 2007]. We discuss in the following various other applications:

1.3.1 Sensor networks

Wireless Sensor Networks (WSN) are a special type of ad hoc networks. They use compact and autonomous devices, called sensor nodes. These nodes collect and transmit their observations autonomously to other nodes or to the central server directly. Sensor networks are used in many applications fields to monitor and to supervise the environment [Liu et al., 2003, Gürgen, 2007]. They are also used in the domain of the electrical energy monitoring [Abdessalem et al., 2007]. Currently, several electrical energy providers are using smart sensors. These latter send in a continuous manner their observations about the users' electricity consumption to the information systems of the electricity suppliers to which they are linked. The recorded data are in the form of streams. The analysis of these data streams makes it possible to detect several anomalies such as the over-consumption of the energy or the failure in a household appliance. PQStream is a data stream management system designed to process and to manage the streams data generated by the Turkish Electricity Transmission System [Küçük et al., 2015]. The system includes a module for continuous data processing, where several data mining methods such as classification and clustering are applied to the data, a database to store the obtained data analysis results, and a Graphical User Interface (GUI).

1.3.2 Financial analysis

The financial analysis is one of the major application of data streams. Previously, the analysis of financial data was intended to assess the probability of a financial crisis of a company. Nowadays, the analysis of these data involves a wide variety of users, including the commercial providers, banks, investors, credit agencies, the stock market, among others. In this context, several data mining operations can be applied to financial data, such as fuzzy logic techniques, machine learning, neural networks, and genetic algorithms [Kovalerchuk and Vityaev, 2000]. The purpose of these operations is to study the impact of one market on another one, monitor the conformity and consistency of the trading operations and improve their performance, and last but not least, trigger warning signs of changing trends [Kovalerchuk and Vityaev, 2000]. For instance, Tradebot [TRA, 1999] is a search engine that allows for quantitative analysis of financial data and trading performance, and the design of strategies and implementation of scientific experiments to evolve the theory of commerce, and ultimately, work with traders to improve the trading system.

1.3.3 Network traffic analysis

Several real-time systems for the analysis of network traffic have been designed. They aim to infer statistics from the network traffic and to detect critical conditions such as congestion and denial of service attacks. GigaScope [Cranor et al., 2002] allows to monitor Internet traffic with an SQL interface. Tribeca [Sullivan and Heybey, 1998] is a stream-oriented DBMS designed to monitor and analyze the network traffic performance. Tribeca has a query language that can be written and compiled by the users to process the data streams coming from the network traffic. Gilbert *et al.* [Gilbert et al., 2001] proposed QuickSAND to summarize the network traffic data using the sketches. In this context, one can search for instance for the clients who consumed the most bandwidth of the network. The analysis of Web traffic has also several interesting applications and serves several purposes [Csernel, 2008]:

- Rank the n most accessed pages during a specific period of time in order to optimize the loading time of these pages.
- Analyze the behavior of the visitors of a particular page or website and identify and determine the distinct users among them.
- Analyze the traffic generated by the social networks.

1.4 Data streams management

Traditional DataBase Management Systems (DBMSs) allow permanent storage and efficient management of the data by exploiting their structure. A query language is used to query the data and retrieve information. However, due to the emergence of data streams, new challenges related to data processing have appeared. These

issues are mainly due to the infinite volume of the stream and its very high arrival rate. These new constraints make the use of the DBMSs inadequate. As a result, the traditional data storage and analysis systems need to be revised to allow processing of data streams. Hence, the emergence of data streams management systems. We detail in the following the main constraints related to the processing of data streams.

1.4.1 Data streams management system characteristics

A DSMS is supposed to meet the constraints of data streams and the needs of the applications that generate these data by having characteristics related to both the functionality and the performance [Gabsi, 2011].

- **Processing continuous queries.** In database applications, the queries are evaluated in a finite environment on persistent data. In such applications, the data does not change as long as the current query is not answered. On the contrary, in streaming applications, data keep growing and the whole environment is fully scalable. As a result, the queries are persistent, they must be executed continuously on volatile data. Also, it is important that the DSMS has a highly optimized engine to handle the volume of data.

- **Data availability.** The DSMS must ensure the availability of the data at all times. It is also supposed to deal with the system failures and must take into account the eventual arrival delay of the data. Due to the eventual long delays in the arrival of data, the operations may be blocked. To avoid such situation, a maximum delay time (time out) can be specified. Thus, queries that can be blocking are processed in a timely manner even if the data is not complete.

- **Infinity of the data.** The DSMS must be able to handle the huge volume of the data stream. The use of the load shedding techniques [Tatbul et al., 2003] and the summaries structures are possible solutions to reduce the load on the system.

- **Resistance to stream imperfections.** The system must handle the imperfections of the data. In the real world, the data often contain noisy, erroneous, duplicate and missing values. The DSMS has to deal with these issues.

1.4.2 Data streams management systems

Several DSMSs have been developed in the recent years. These systems are distinguished by the query reformulation languages, the procedures used to represent the streams, and the type of application they are designed for. Thus, some DSMSs have a generalist vocation, such as Aurora [Abadi et al., 2003], TruViso [TRU, 2004], Medusa [Cetintemel, 2003], Borealis [Abadi et al., 2005], TelegraphCQ [Chandrasekaran et al., 2003], and StreamBase [Tatbul et al., 2003], while others are intended for a particular type of application, such as GigaScope [Cranor et al., 2002], NiagaraCQ [Chen et al., 2000], OpenCQ [Liu et al., 1999], StatStream [Zhu and Shasha, 2002] and Tradebot [TRA, 1999].

One of the well-known DSMSs is STREAM (STanford stREam datA Manager) [Arasu et al., 2016]. Akhtar *et al.* [Akhtar, 2011] evaluated the performance of the STREAM system and noted several advantages of it, namely, that it is user friendly since it allows the users to interact with the system through a graphical interface, and that it is very suitable for the applications that require high precision since it gives accurate results for the aggregation queries. We present in the following the STREAM system.

STREAM: STanford stREam datA Manager

STREAM [Arasu et al., 2016] is a general DSMS developed in C ++ language at the Stanford University. The users can register their queries and receive the corresponding results as a streaming HTTP response in XML format using a Web-based GUI through direct HTTP. This DSMS is based on the Continuous Query Language (CQL) declarative language derived from SQL, in order to formulate continuous queries on relations (static data) and data streams (dynamic data). The semantic of continuous queries on relations and data streams relies on abstract relational semantics. This semantic is based on two types of data, streams and relations defined as follows [Arasu et al., 2016]:

- A stream S is an unbounded set of pairs $\langle s, \tau \rangle$, where s is a tuple and τ is the logical arrival time of tuple s on stream S.
- A relation R is a time-varying set of tuples, where R(τ) is a relation representing the set of tuples at time τ.

This semantic uses three blocks of operators:

- A relational query language, which we can see as a set of relation-to-relation operators.
- A window specification language. It can be seen as a set of stream-to-relation operators to convert the streams into relations. These operators are based on the sliding window model and are expressed using a window specification language derived from SQL-99. The sliding window can be of three types: a tuple-based sliding window, a time-based sliding window, and a partitioned sliding window.
- A set of relation-to-stream operators: Istream which is applied to a relation R whenever a tuple s is inserted into the relation at time τ, Dstream applied to a relation R whenever a tuple s is deleted from the relation at time τ, and Rstream applied to a relation to send all the tuples s belonging to a relation R at the time τ.

An example of a CQL continues query on a data stream *flow_meter* is as follows:

```
SELECT Istream(timestamp, consumption) FROM flowmeter [ROWS 96]
WHERE flowmeter.consumption > 500
```

This query answers the following question: In the last 96 observations, what are the instants during which the quantity of the water consumed by the flow_meter has exceeded the value of $500m^3$.

1.5 Sampling algorithms

In streaming environments, the data arrive continuously, often at a high rate, and the system that receives the data may not have a sufficient memory to store them exhaustively. Thus, data stream processing implies reducing the size of the data by maintaining and storing a summary of the data in the memory. Sampling algorithms are used to construct a data stream summary. An effective summary of a data stream must have the ability to respond, in an approximate manner, to any query whenever the time period investigated. The purpose of the sampling algorithms is to provide information concerning a large set of data from a representative sample extracted from it. Data streams sampling algorithms are based on the traditional sampling techniques. These techniques require accessing all the data in order to construct the sample, also called summary. However, in streaming context, this condition is not guaranteed because of the infinite size of the stream. Thus, the sampling algorithms have to be adapted to the streaming context using the windowing models presented in Section 1.2.3.

In the following, we present in detail the different sampling algorithms proposed in the literature to construct a data stream summary.

a. **Simple Random Sampling (SRS).** The SRS algorithm [Cochran, 1977] is the most used sampling algorithm. It is simple and gives a random sample. It consists of sampling the data in a random manner, where each item of the data has the same probability p of being selected. Data sampling can be with or without replacement. With the SRS with replacement, the sample may contain duplicate elements since each item of the data may be selected twice or more. Whereas, with the SRS without replacement, each item can be selected only once which makes this type of sampling more accurate and convenient.

b. **Systematic Sampling.** Let n be the sample size and N the size of the data, Systematic sampling algorithm divides the data into n groups, each one of size $k = N/n$. Then, it chooses a random number $j \in [1, k]$ and adds the following elements to the sample: j, $j + k$, $j + 2k$, $j + 3k$... [Cochran, 1977]. Systematic sampling algorithm has several advantages, the sample is easy to be built, it is faster and more accurate than Simple Random Sampling algorithm since the sampled elements are spread over the entire data [Cochran, 1977]. One drawback of this algorithm is its lack of randomness in the sample. It fact, the sampled elements are periodically selected which can impact the quality of the sample. If the original dataset presents a periodicity close to the value k, the sample cannot represent the original dataset, and it will be biased in such case.

c. **Stratified Sampling.** Stratified sampling algorithm [Cochran, 1977] is a Simple Random Sampling where the original data is divided into homogeneous

subgroups, called strata, according to one or more predefined criteria. A sample will then be extracted from each subgroup by applying the Simple Random Sampling algorithm. Through stratification of the data, the stratified sampling algorithm reduces the sampling error and ensures a high level of representativity of the sample compared to the Simple Random Sampling algorithm. The more the groups are heterogeneous with each other and homogeneous internally, the more the sampling accuracy is high. The use of this type of sampling is beneficial in several cases, for example, when it is desired to highlight a specific subgroup within the data and ensure its presence in the sample. Stratified sampling is also used to represent the smallest, extreme or rare subgroups of the data in the sample. Since each element of the original data must be associated with a subgroup beforehand the sampling, the construction of the sample using the Stratified sampling algorithm will be more expensive than that with the Simple Random Sampling algorithm. The choice of the sample size within each stratum and the choice of the number of strata are the principal issues encountered with the Stratified sampling algorithm.

d. **Weighted Random Sampling (WRS) without replacement.** A general summary must be representative of the entire stream. In some cases, this condition is not satisfied. In fact, some data may be over-represented or under-represented in the sample. Subsequently, the statistical inferences and conclusions drawn from this sample will be unreliable. This problem is known as "non response" in the survey theory. In such a situation, a correction of the sample is suggested to overcome the lack of representativeness of the sampled data. One of the correction solutions is the weighting of the survey. In contrast to the SRS where all the data have the same probability of being included in the sample, the WRS samples each item with a probability that depends on its associated weight [Efraimidis, 2015].

   Efraimidis *et al.* [Efraimidis, 2015] introduced two types of WRS: WRS-N-P and WRS-N-W. Their difference lies in the way of calculating the sampling probability of the data. WRS-N-P is the WRS without replacement with defined probabilities. The sampling probability of each item e_k is proportional to the relative weight w_k of the item, and is calculated as follows:

$$p_k = \frac{\alpha \times w_k}{\sum_{i=1}^{k} w_i} \tag{1.1}$$

   where α is the size of the sample.

   WRS-N-W is the WRS without replacement with defined weights. The sampling probability of each item e_k is proportional to its relative weight w_k with respect to the weights of all not sampled items. The sampling probability p_k of the item e_k is therefore calculated as follows:

$$p_k = \frac{w_k}{\sum_{i \in v-S} w_i}$$

   where S represents the sample.

   A-Chao [Chao, 1982, Efraimidis, 2015] is a reservoir-based WRS algorithm without replacement and with defined probabilities. It proceeds as follows to

construct and to maintain a sample of fixed size k: the first k items of the stream are added to sample with a probability $p = 1$. For each new incoming item, A-Chao adds it to the sample with the probability given by the equation 1.1 while deleting randomly another item from the sample. The acceptance/rejection algorithm [Olken and Rotem, 1992] samples each item of the data with a probability proportional to its weight. The weights are associated in a random manner to the items. To construct a random sample of size $k = 1$, the algorithm generates a random number $j \in [1, n]$ and samples the item with index j with a probability equal to $\frac{w_j}{w_{max}}$ where w_{max} is the maximum weight among all the items. If the item j is not sampled, the process will be repeated until an item is selected. A-Res presented in [Efraimidis and Spirakis, 2006] is a reservoir-based WRS algorithm without replacement. It maintains a sample of fixed size k. Firstly, it adds the first k items to the sample with a probability $p = 1$. Then, for each sampled item, a key is calculated as follows:

$$key_i = {u_i}^{1/w_i}$$

where u_i is a random value $\in [0, 1]$. After that, for each new incoming item e, calculate its key and find the smallest key l in the sample. If the key of the item e is greater than l, replace the item having the smallest key in the sample by the item e. The main concern with these three algorithms is the lack of a policy for the definition and the updating of the weights. The weights are generated in a random manner and only once.

e. **Reservoir sampling.** The goal of the Reservoir sampling algorithm [Vitter, 1985] is to maintain a uniform random sample of a fixed size k from the entire data stream, without requiring a priori knowledge of the stream size. Firstly, the algorithm adds the first k received elements of the stream to the reservoir (sample), each with a probability equal to 1. After that, with the arrival of new elements, the algorithm adds each element e to the sample with a probability $p = k/i$, where i is the index of e, while deleting a random element from the sample. This algorithm is simple and suitable for the streaming environment as it is executed in one pass. The main concern with the reservoir sampling algorithm is that the sample becomes irrelevant over time. In fact, the more recent the elements, the less likely they are to be included in the sample.

f. **Backing sampling.** Backing sampling [Gibbons et al., 1997, Gibbons et al., 2002] is a uniform random sampling algorithm based on the Reservoir sampling algorithm [Vitter, 1985]. It was designed to overcome the problems of expired data in a sliding window, which are not handled by the reservoir sampling algorithm. The algorithm starts by adding the first k received elements of the stream to the reservoir with a probability equal to 1. Then, the algorithm skips a random number of elements and add the next element to the reservoir with a probability equal to the sampling rate. Another random number of items are skipped, and so forth. The aim of backing sample algorithm is to maintain a sample containing only the unexpired elements of the stream. For this purpose, two bounds are defined: the upper bound K representing the maximal size of the reservoir, and the lower bound L representing the minimal size of the reservoir. When an element expires, the algorithm removes it from the sample if it was present. Successive deletions of the expired elements lead

to the decrease of the size of the reservoir. Therefore, at each time the size of reservoir becomes smaller than the lower bound, the sampling process will be reinitialized and a new reservoir containing k elements will be reconstructed.

g. **Concise sampling.** Gibbons *et al.* [Gibbons and Matias, 1998] proposed the Concise sampling algorithm to construct and to maintain a concise representation of the data, where each element that occurs several times in the original data will be represented in the sample as a pair $\langle value, count \rangle$, where $value$ is the value of the item, and $count$ in the number of occurrence of this item in the original data. If an item occurs only once, it will be represented as a singleton $value$. The concise sampling algorithm defines a $footprint$ for the sample which represents the size of the sample to be stored in the memory. This size is defined as the number of the items in the sample and the corresponding $count$ value. At first, all the received items are added to the sample with a probability $p = 1$. As new items arrive, the $count$ value of each item in the sample will be updated, and the new incoming items will be added randomly to the concise sample. If the size of the sample exceeds the predefined $footprint$, the algorithm proceeds to reduce the size of the sample either by deleting singleton items in a random manner, or by decreasing randomly the $count$ values of the items in the sample.

h. **Chain-sample.** The main difficulty in sampling over a sliding window is the expiration of elements. In fact, expired elements must be replaced in the sample in case they are present. Assuming that recent data are more important than older data, Chain-sample [Babcock et al., 2002] maintains a sample of fixed size $k = 1$ over a logical sliding window. For a sample of size $k > 1$, the algorithm is repeated k times. At the first stage, the algorithm selects an element e_i from the first window with a probability $Min(i, n)/n$, where i is the index of e_i in the stream and n is the size of the window. Then, a random replacement element r is selected from the group of elements with indexes going from $i + 1$ to $i + n$ and it will replace e_i when this latter expires. When r arrives at the current window, it will be stored and a random replacement element is chosen from the elements going from $r + 1$ to $r + n$, and so on. Chain-sample algorithm has the disadvantage of generating a sample containing duplicates elements in case $k > 1$.

i. **Priority sampling.** In addition to the Chain-sample algorithm, Babcock *et al.* [Babcock et al., 2002] proposed the Priority sampling algorithm to construct and to maintain a uniform sample over time-based sliding windows. The key idea of the Priority sampling algorithm is to assign a random priority $p \in [0, 1]$ for each incoming item of the stream and to select the item having the highest priority in the window.

j. **Random Pairing (RP) sampling.** The RP sampling algorithm [Gemulla et al., 2006, Gemulla, 2008] builds and maintains a uniform sample of fixed size over a sliding window. RP retains three measures on each window: c_1 which represents the number of expired items that are included in the sample, c_2 which counts the number of expired items that are not included in the sample, and d which depicts the number of all expired items: $d = c_1 + c_2$. At each time a sampled item expires, RP removes it from the sample. When a new item arrives in the window, it can be added to the sample depending on the value of

d. If $d = 0$ (the expired element in the window is not included in the sample), the addition of the new item to the sample follows the Reservoir sampling algorithm [Vitter, 1985]. On the contrary, if $d > 0$, the new item will be added to the sample with a probability equal to $\frac{c_1}{c_1+c_2}$. Following this step, the values of c_1, c_2 and d are updated.

k. **StreamSamp.** StreamSamp [Csernel, 2008] is a progressive sampling algorithm based on the Simple Random Sampling algorithm. As soon as they are received, the items of the stream are sampled with a fixed sampling rate p. When the predefined sample size k is reached, StreamSamp associates the order 0 to the sample and stores it, and constructs a second sample of the same size k, and so on. As the size of the stream increases, the number of samples of order 0 also increases. When this number exceeds a given bound, StreamSamp proceeds to fuse the two old samples of order 0 into one sample of size k by performing a Simple Random Sampling of rate $p = 0.5$. The new obtained sample is of order 1, and so on.

l. **Distance-based Sampling for Streaming data (DSS).** DSS [Dash and Ng, 2006] maintains a sample of a fixed size k and updates it each time a new element is added to the stream. The algorithm manages the insertions into the sample so that the difference between the sample and the stream is minimal. This difference is defined by $\sum_{A \in I} |f(A, S_0) - f(A, S)|$ where I denotes the set of frequent item sets. Initially, the first k items are added to the sample S_0 with a probability equal to 1. DSS uses the notion of ranking to manage the insertions in the sample. The element with the highest ranking is the one that its deletion from the sample engenders an important increase of the difference between S_0 and S. On the contrary, the element with the lowest ranking denoted LRT, is the element of the sample whose presence or absence from the sample will almost not affect the difference between S_0 and S. Initially, two tables for the ranking of the elements contained in both S_0 and S are initialized. The classification of the elements in each of the two tables is calculated from the equation:

$$Dist = Dist(S_0 - t, S)$$

where t is the element.

When a new element t is added to the stream, two distances will be calculated: $D_{without}t = Dist(S_0, S + t)$ and $D_{with}t = Dist(S_0 + t - LRT, S + t)$. If $D_{without}t > D_{with}t$, the presence of t in S_0 is favored, and it will replace LRT in the sample. If $D_{with}t > D_{without}t$, the element will be skipped. DSS is very expensive, each time a new element is added to the stream, the distance $Dist$ has to be calculated and the set of frequent item sets of the entire stream must be again computed.

Performance metrics

The quality of a summary depends on the sampling algorithm used to construct it. We propose in the following to evaluate the effectiveness of a data stream sampling algorithm according to the following metrics:

- **Number of passes over the data.** One of the main constraints that a data stream sampling algorithm must satisfy is the single pass on the data. In fact, since it is impossible to store all incoming data for further processing, the data stream has to be treated on the fly and without prior storing the entire data [Muthukrishnan et al., 2005]. Hence, it is important to make sure that the sampling algorithm makes only one pass over the data to sample them. The complexity of the sampling algorithm to deal with each item of the stream is of $O(1)$, independent of the size of the whole stream.

- **Memory consumption.** Data stream is by definition infinite. Most sampling algorithms use a continuous increasing sample size. The sample size is often proportional to stream size. It depends on the sampling rate which is set according to the required accuracy. With a high sampling rate, very few information about the original dataset will be lost. However, this requires more resources, in particular, memory usage to store the sample. Some other sampling algorithms (such as Reservoir sampling [Vitter, 1985] and StreamSamp [Csernel et al., 2006]) use a fixed bounded memory independent of the stream size. In this case, the sample is always updated to replace old elements.

- **Skewing ability.** The skewing ability is the possibility to give more chance for some particular items to be selected and added to the sample. It can be based on the content of the item or its timestamp. For example, a water network explorer can execute the following query to check if a high water consumption has occurred in a given period of time.

```
SELECT timestamp, consumption
FROM flowmeter
WHERE consumption > 500
AND timestamp > '01/01/2017'
AND timestamp <= '07/01/2017';
```

  Therefore, it is compulsory to force the selection of this data (if it is present among the data) during the sampling process, by giving it a greater weight compared to other data (note that a sampling algorithm without a skewing ability, implicitly associates a weight for all data). Missing such data because of the sampling will engender a false answer to the considered query, and an important information to the network explorer will be consequently lost. Besides, the skewing of the data must comply with a certain well-defined policy. This latter will define how to associate the weights, and how to calculate and update them over time according to the objective of the application.

- **Resources consumption.** The sampling algorithm has to be fast enough to deal with the high rate of the current data streams. Therefore, a low complexity is required to reduce the execution time and the CPU charge of the sampling algorithm. This criterion is particularly important when implementing the sampling algorithm on devices with limited resources such as sensors. In fact, in WSN, the sensors are devices with limited resources in terms of battery power, CPU, memory, and network bandwidth. Because of the memory limitations, the WSN devices cannot store a lot of information. However, each node must compute, process and send its observations to other nodes or to the central

system. Therefore, reducing the resources requirements is indispensable [Jain and Chang, 2004]. Implementing low complexity sampling processes on WSN devices is a solution to conserve the energy consumption and the CPU usage and to deal with the memory and time constraints [de Aquino et al., 2007].

1.6 Discussion

In this chapter, we presented the basic concepts of data streams. We focused on the sampling algorithms used for the construction of general summaries in streaming environments. A brief comparison of these algorithms is presented in Table 1.3. Four metrics are considered to study the effectiveness of the sampling algorithms: number of passes over the stream, skewing ability, memory and resource consumption. The choice of the sampling algorithm is closely dependent on the constraints of the targeted application. Based on the needs and the preferences of the user regarding the age of the data and the period of time addressed, the fixed, sliding or landmark windowing model will be adopted. The choice of a weighted sampling algorithm depends on the user's preferences regarding the sample's content. When the user prefers to give more importance to certain data than others, and therefore, to force their sampling, weighted sampling algorithms are recommended. Finally, depending on the memory and resources consumption constraints, it can be preferred to use a sampling algorithm that provides a bounded/unbounded sample size.

We identify several research challenges that will shape our work in the next chapters. Specifically, we will study in details in Chapter 2 the choice of the sliding window size and the sample size since they have both an important impact on the computational resources required by the sampling algorithm.

TABLE 1.3 – Weaknesses of data streams sampling algorithms.

Sampling Algorithm	Weak Points
Simple Random [Cochran, 1977]	Unbounded sample size No policy for old data replacement No skewing ability
Systematic [Cochran, 1977]	Unbounded sample size Biased sample in case of periodicity in the data stream No skewing ability
Stratified [Cochran, 1977]	Unbounded sample size How to choose the sample size? No skewing ability
Weighted Random [Efraimidis and Spirakis, 2006, Efraimidis, 2015]	Unbounded sample size No policy for the determination and revision of the weights
Reservoir [McLeod and Bellhouse, 1983, Vitter, 1985]	Recent elements have less chance of being sampled No skewing ability
Distance-based [Dash and Ng, 2006]	Very expensive in terms of execution time Performs several passes over the data No skewing ability
Concise [Gibbons and Matias, 1998]	No skewing ability
Backing [Gibbons et al., 1997, Gibbons et al., 2002]	Performs several passes over the data No skewing ability
Chain [Babcock et al., 2002]	Redundant data in the sample No skewing ability
Priority [Babcock et al., 2002]	No policy for the determination and revision of the weights
Random Pairing [Gemulla et al., 2006, Gemulla, 2008]	No skewing ability
StreamSamp [Csernel, 2008]	No policy for the determination and revision of the weights

Chapter

2 Chain-sample algorithm for data streams sampling over a sliding window

Contents

2.1 Introduction **32**

2.2 The traditional Chain-sample algorithm **32**

2.2.1 Motivation 32

2.2.2 Algorithm description 33

2.2.3 Collision problem 33

2.3 Chain+: A free redundancy Chain-sample algorithm **35**

2.3.1 Construction of the samples 35

2.3.2 Memory usage for a single *chain-sample* 36

2.3.3 Trade-off between the execution time, sampling rate and window size 38

2.3.4 Comparison of the Chain+ sampling against the Simple Random Sampling algorithm 39

2.4 Enhancing the Chain+ sampling algorithm **42**

2.4.1 Inverting the selection for high sampling rates strategy . . 42

2.4.2 Divide-to-Conquer strategy 43

2.4.3 Experimentations 45

2.5 Conclusion **46**

The scientific contribution and the obtained results presented in this chapter have been published in our paper: **Rayane El Sibai**, Yousra Chabchoub, Jacques Demerjian, Zakia Kazi-Aoul and Kablan Barbar. *"A performance study of the Chain sampling algorithm"*. In the 7^{th} International Conference on Intelligent Computing and Information Systems (ICICIS), pages 487-494, IEEE, December 2015.

2.1 Introduction

Online analysis of data streams is a major challenge because of the ever-increasing rate of data streams coming from heterogeneous sources and belonging to different application fields. To reduce the volume of a given data stream, several sampling methods have been designed by the research community. When sampling over a sliding window, the main difficulty is how to maintain and update online a representative sample of the window over time.

We study in this chapter the Chain-sample algorithm. The purpose of this algorithm is to select, randomly, and at any time, a fixed proportion of the data among the most recent elements of the stream contained in the last sliding window. We show in this chapter that the Chain-sample algorithm has some drawbacks mainly due to the collision problem. The collision occurs when the same element is selected to be included in the sample more than once during the execution of the algorithm. We propose two approaches to overcome this weakness and to improve the Chain-sample algorithm. The first one is called "Inverting the selection for high sampling rates" strategy and the second one is inspired by the "Divide-to-Conquer" strategy. Various experiments are carried out to show the effectiveness of these two improvements, in particular, their impact on the execution time of the algorithm.

This chapter is organized as follows. We discuss in Section 2.2 the Chain-sample algorithm and its weakness. We present in Section 2.3 a new version of the algorithm, called Chain+, for which we conduct an in-depth study. We propose in Section 2.4 two improved versions of the Chain+ sampling algorithm, and we validate our improvements by simulation. We end the chapter with a conclusion.

2.2 The traditional Chain-sample algorithm

2.2.1 Motivation

In several fields of data streams applications, the data are time-sensitive, the most recent data are more important and relevant for the application than the historical data of the stream. Examples of such applications include the network supervision [Cormode, 2013], sensors networks [Carney et al., 2002], social media [Osborne et al., 2014], network traffic management [Babu et al., 2001], and others. In order to promote the recent data of the stream, the range of the queries can be restricted to a sliding window. Therefore, to obtain an approximate response to a query, only the data in the last sliding window of the stream will be evaluated.

When sampling over a sliding window, the principal challenge is to maintain online a representative sample of the window over time. Actually, the sample must be continuously updated by taking off and replacing the expired elements. The Chain-sample algorithm introduced by Babcock *et al.* in [Babcock et al., 2002] assumes that the recent data are more important than older data, and maintains a sample of fixed size k over a tuple-based sliding window of size n. In comparison with the other data streams sampling algorithms, the Chain-sample algorithm has

the following advantages:

- It performs a single pass on the data stream, which makes it suitable for streaming environments.
- By using the sliding window model, the Chain-sample algorithm focuses on the recent elements of the stream while ignoring the old elements.
- No prior knowledge about the stream is required.
- It provides a random and uniform sample selected from the n most recent elements of the stream.
- It uses a small and bounded memory.

2.2.2 Algorithm description

Assuming that the recent data of the stream are more important than the historical data, the purpose of the Chain-sample algorithm [Babcock et al., 2002] is to provide, at any time, a uniform random sample of exactly k elements from the most recent n elements of the stream. It uses a tuple-based sliding window to give more importance to the recent elements of the stream. We assume that the stream is composed of elements with an ever increasing index, to provide a sample of size k, the Chain-sample algorithm maintains k independent samples over each window. Each sample is constructed as follows: In the first window of n elements, each item is selected with a probability equals $\frac{min(i,n)}{n}$, as soon as it arrives, where i is the index of the item in the window. Once chosen, a successor's index j for the i^{th} item is chosen randomly among the items with indexes $\in [i+1, i+n]$. When the item with index j arrives at the window, it will be saved in the memory and a random successor for it will be chosen with the same method as for the item i. When the item with index i expires, it will be removed from the sample and replaced by its successor j. These steps are detailed in Algorithm 1.

Figure 2.1 illustrates an example of selecting $k = 1$ element from a sliding window of size $n = 4$ using the Chain-sample algorithm.

2.2.3 Collision problem

When the sample size k is greater than 1, the Chain-sample algorithm builds k independent samples, each of size $k' = 1$. It proceeds then to merge the k samples to construct the final sample of the window. This process gives rise to the collision and redundancy problems. Indeed, following this process, the current and also future samples of the upcoming windows will contain each a number of duplicated elements r, $0 \leq r \leq k$. This will lead over time to the degradation of the quality of the samples. We refer to this scheme as a Chain-sample algorithm with replacement. We define the quality of the sample of a window as follows:

Definition 1. *The quality of a sample is defined as the ratio between the number of distinct sampled elements and the total number of sampled elements.*

Initially, the collision problem starts on the first window of the stream, from which the Chain-sample algorithm selects k elements in an independent manner.

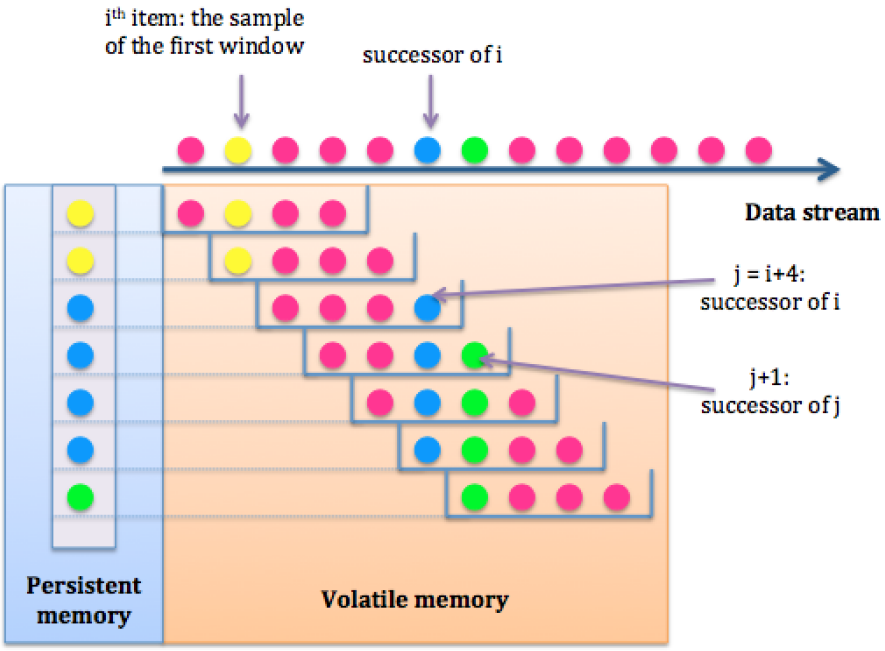


FIGURE 2.1 – Sampling $k = 1$ item over a sliding window of size $n = 4$, using the Chain-sample algorithm.

Algorithm 1. Chain-sample algorithm with replacement over a purely sliding window

procedure CHAINSAMPLING(k, n) ▷ k is the sample size, n is the window size
Initialization: S_w is the sample for the window w, at the beginning $S_1 = \varnothing$
  repeat
    for $i \in [1, n]$ **do**
      Add e_i to S_w with a probability $p = \frac{min(i,n)}{n}$
      Select a random successor's index r to replace e_i, $r \in [i+1, i+n]$
    end for
  until selecting k items
  while a new item e_i is received, $(i > n)$ **do**
    $w \leftarrow w + 1$ ▷ move the window by one step
    $j \leftarrow i - n$ ▷ j is the index of the expired element
    Add the items of S_{w-1} to S_w
    if $e_j \in S_{w-1}$ **then**
      Remove the item e_j from S_w
      Add the replacement e_z of e_j to S_w
      Select a random index $y \in [z+1, z+n]$, e_y will be the replacement of e_z
    end if
  end while
  return S_w
end procedure

The collision problem remains in the future windows of the stream since two or more items can choose the same successor item. In order to construct a sample free of redundancy, the selected k elements on each window must be distinct. However, this prerequisite is not always satisfied since it relies narrowly on two parameters: the

size of the sample k and the size of the window n.

Figure 2.2 depicts the experimental collision rate obtained with the traditional version of the Chain-sample algorithm as well as the theoretical probability of collision $P_{collision}$ for selecting k items among n items using the SRS algorithm with replacement. Several sampling rates k/n are considered and the size of the window is fixed to 10 items.

The theoretical probability of collision $P_{collision}$ is given by the following equation:

$$\begin{aligned} P_{collision} &= 1 - P_{Selecting\ k\ distinct\ items} \\ &= 1 - \frac{n!}{n^k(n-k)!} \end{aligned}$$

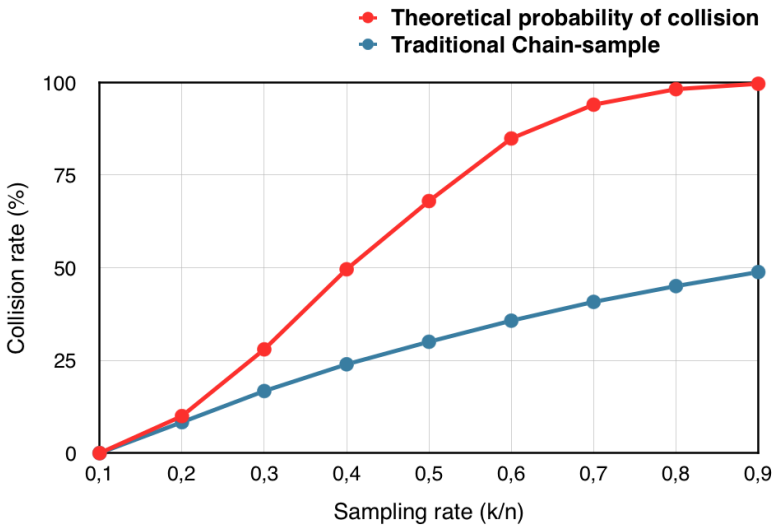


FIGURE 2.2 – Collision rate.

As shown in Figure 2.2, 25% of the sampled elements are duplicated when the sampling rate is equal to 50%, which has a significant impact on the quality of the sample. Also, one can notice that the collision rate with the traditional version of the Chain-sample algorithm is lower than that of the SRS algorithm with replacement. This can be explained by the fact that, with the Chain-sample algorithm the successors of the k elements are not selected from the same window. For each selected item with index i, its successor belongs to the window $[i+1, i+n]$. The k windows of size n can partially overlap, but they are distinct which reduces the risk of collisions.

2.3 Chain+: A free redundancy Chain-sample algorithm

2.3.1 Construction of the samples

The quality of the samples built on the windows of the stream depends on two factors: the quality of the basic sample and the procedure for selecting the replacements.

These two factors are independent of each other. In fact, whether the elements of the basic sample are distinct or not, the redundancy problem occurs when selecting the replacements since two or more elements can choose the same successor. In order to build and to maintain a sample of size equal to k, we propose to force the Chain-sample algorithm to select and to preserve a sample of k distinct elements on each window of the stream. Therefore, we modify the building procedure of the basic sample as well as the selection procedure of the replacements. We give in the following the definition of the basic sample.

Definition 2. *The basic sample is the sample built on the first window of the stream.*

To build a sample of size k, the Chain+ sampling algorithm constructs a single sample of size k on the first window of the stream instead of building k independent samples each of size $k' = 1$. Each element of index i in the first window is chosen with a probability equals $\frac{min(i,n)}{n}$ and if and only if it is not already present in the basic sample. Otherwise, another element will be chosen with the same probability, and so on. This process will be repeated until choosing k distinct elements. This alteration guarantees that the sampled elements in the basic sample are distinct, however, it does not ensure that the future samples will be free of redundancies. To adjust this problem, we propose to maintain a list of replacements over time. This list contains only the indexes of the replacements associated with the sampled elements on each window. Thus, before picking a replacement, the Chain+ sampling algorithm checks whether the replacement is present in the list. If it is the case, another replacement will be randomly selected. Since the size of the stream is infinite, and as it is not necessary to store the indexes of all the replacements from the beginning of the stream, the list of replacements is periodically updated as follows: for each new element e_i of the stream, all the replacements with indexes smaller than i are fired.

The detailed pseudo code of the Chain+ sampling algorithm is presented in Algorithm 2.

2.3.2 Memory usage for a single *chain-sample*

We recall that when an item is sampled, a random successor will be associated with it. When this successor arrives at the current window, it will be stored in the memory and a successor will be selected for it, and so on, thus, building a chain of elements, called *chain-sample*. The size of a single *chain-sample* is defined as follows:

Definition 3. *The size of a single chain-sample is given by the sampled element plus the number of successors associated with that element.*

Assuming that the element with index i is the oldest element in the sample of the current window, the size of the *chain-sample* of the item i is given by the following equation [Babcock et al., 2002]:

Algorithm 2. Chain+ sampling algorithm: Chain-sample algorithm without replacement over a purely sliding window

procedure CHAIN+SAMPLING(k, n) ▷ k is the sample size, n is the window size
Initialization: S_w is the sample for the window w, at the beginning $S_1 = \varnothing$
  repeat
    for $i \in [1, n]$ **do**
      if $e_i \notin S_w$ **then**
        Add e_i to S_w with a probability $p = \frac{min(i,n)}{n}$
        repeat
          Select a random successor's index r to replace e_i, $r \in [i+1, i+n]$
          Add r to the list of replacements Rep
        until selecting a successor that does not exist in Rep
      end if
    end for
  until selecting k distinct items in S_w
  while a new item e_i is received, $(i > n)$ **do**
    $w \leftarrow w + 1$ ▷ move the window by one step
    $j \leftarrow i - n$ ▷ j is the index of the expired element
    Add the items of S_{w-1} to S_w
    if $e_j \in S_{w-1}$ **then**
      Remove the item e_j from S_w
      Add the replacement e_z of e_j to S_w
      repeat
        Select a random index $y \in [z+1, z+n]$, e_y will be the replacement of e_z
      until selecting a successor that does not exist in Rep
      Add the successor's index y to the list of replacements Rep
    end if
  end while
  return S_w
end procedure

$$T[1] = 1$$

$$T[i+1] = 1 + \frac{1}{n}\sum_{j=1}^{i} T[j]$$

According to [Babcock et al., 2002], the size of the window has no impact on the size of the *chain-sample*. In order to verify this result, we study the size of the *chain-sample* over time. We choose $k = 1$ and we consider different values of n. For each window size n, the average size of the *chain-sample* is calculated as a mean over 10 iterations. The results are plotted in Figure 2.3. They confirm the theoretical result regarding the independence between the window size n and the size of a *chain-sample*.

Other theoretical results about the length of a *chain-sample* were stated by [Babcock et al., 2002]. They are about the expected bound and high probability upper bound of the length of a single *chain-sample*. To check these results, we

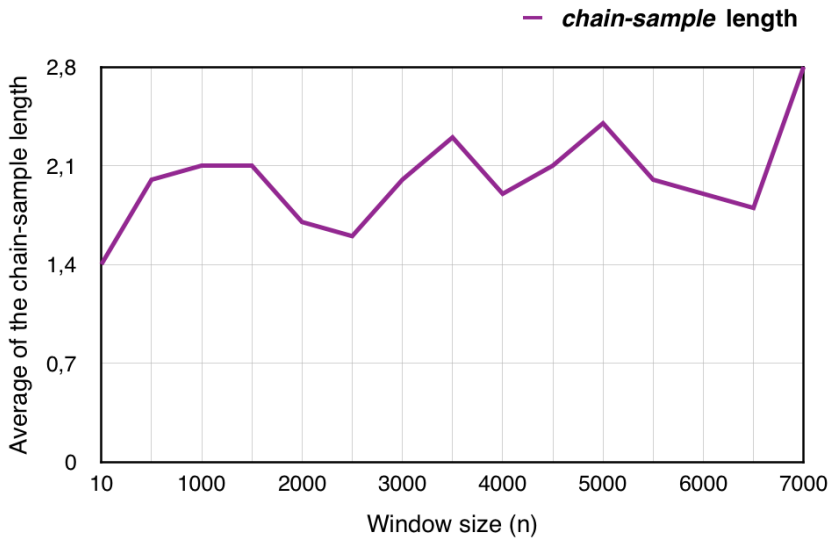


FIGURE 2.3 – Impact of the window size on the *chain-sample* length.

examine the variation of the length of a single *chain-sample* over time, and we fix the size of the window to 1000 items. Figure 2.4 depicts the obtained results. One can easily notice that the average length size of a *chain-sample* exceeds sometimes the theoretical upper bound $e = 2.72$, nevertheless, it is almost always below the high probability upper bound $log(n)$.

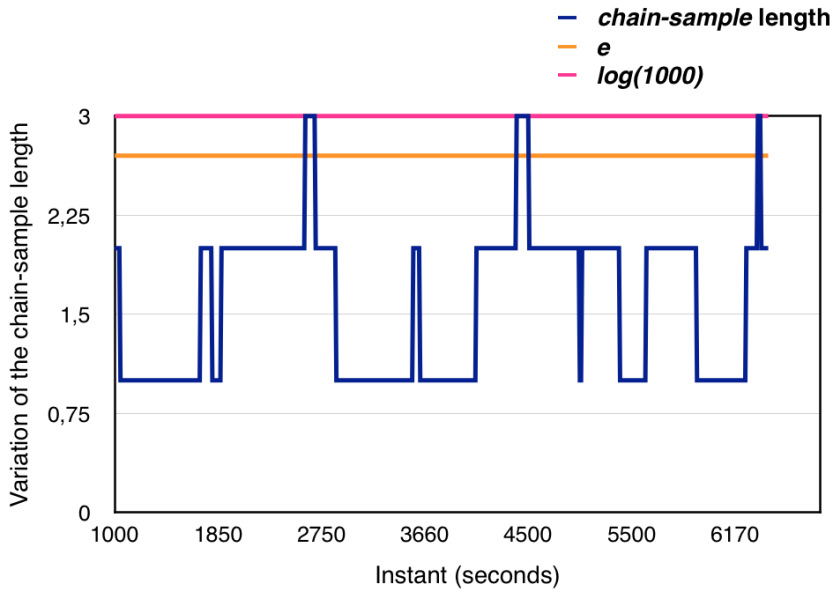


FIGURE 2.4 – Variation of the *chain-sample* length over time.

2.3.3 Trade-off between the execution time, sampling rate and window size

As explained before, when performing the Chain-sample algorithm, if the sample size k is higher than 1, two or more *chain-samples* can include the same items with the same indexes. Therefore, the so obtained k samples may include many redundant

items. To obtain exactly k distinct samples, the indexes of the k samples must be different. One solution to overcome this problem is to check if the successor is already chosen by another *chain-sample* while selecting it randomly. If it is the case, one has to choose randomly another successor. Despite the fact that this solution does not affect the memory consumption of the algorithm, however, it adds a significant overhead in terms of execution time which increases dramatically when the sample size is close to the size of the window.

Figure 2.5 illustrates the impact of the window size n on the execution time of the Chain+ sampling algorithm according to the window size. According to this Figure, one can notice that the window size has no impact on the execution time when the sampling rate is ≤ 0.5. However, for a sampling rate > 0.5, the window size has a noteworthy impact on the execution time which increases with the increase of the window size. Figure 2.6 depicts the impact of the sampling rate k/n on the execution time of the Chain+ sampling algorithm. Several window sizes n are considered. The obtained results show that when the window size is high (≥ 1000), the sampling rate has a considerable impact on the execution time which increases with the increase of k/n.

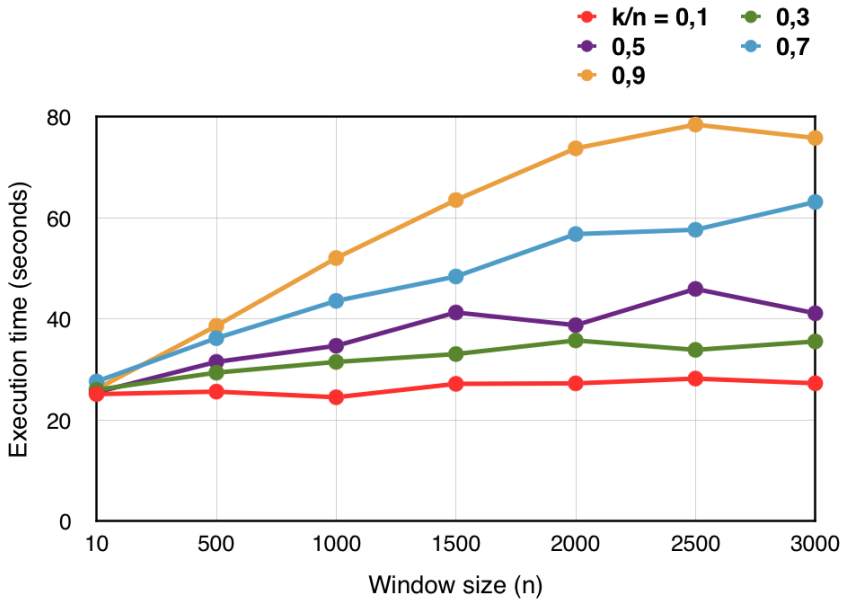


FIGURE 2.5 – Impact of the window size on the execution time of the Chain+ sampling algorithm, for different sampling rates k/n.

2.3.4 Comparison of the Chain+ sampling against the Simple Random Sampling algorithm

The SRS algorithm is the most used sampling algorithm. It consists of selecting each element with the same probability $p = k/n$. It is simple and gives an unbiased sample. However, in its basic version, it does not use the sliding window model. In order to compare the performance of the SRS algorithm to that of the Chain+ sampling algorithm, we adapt SRS algorithm to the sliding window context as follows:

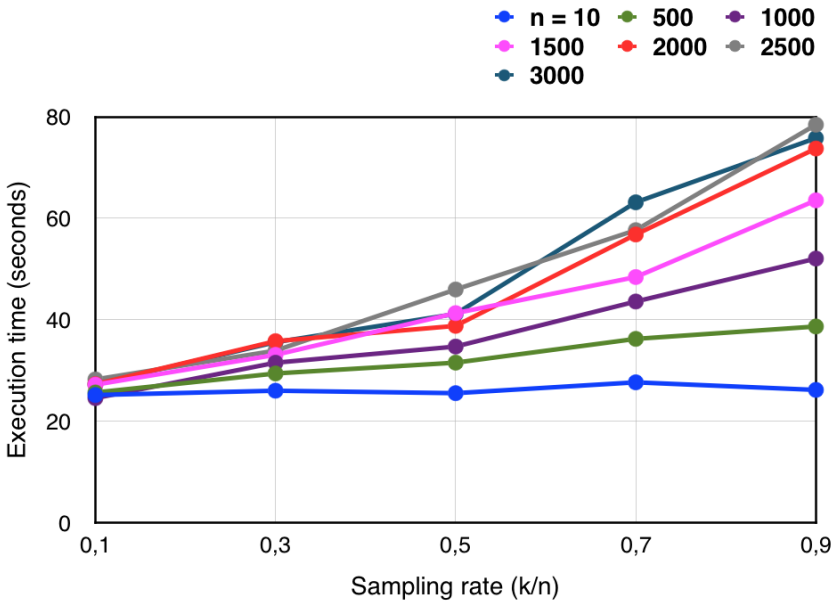


FIGURE 2.6 – Impact of the sampling rate on the execution time of the Chain+ sampling algorithm, for different windows sizes n.

For each received item of the stream:

- Update the current sample by removing the expired elements.
- Add the new received item to the current sample with a probability $p = k/n$.

The detailed pseudo code of the SRS algorithm without replacement over a purely sliding window is given in Algorithm 3.

Algorithm 3. Simple Random Sampling without replacement over a purely sliding window

procedure SRS(k, n) ▷ k is the sample size, n is the window size
Initialization: S_w is the sample for the window w, at the beginning $S_1 = \varnothing$
  while a new item e_i is received **do**
    if $i \leq n$ **then** ▷ in the first window
      Add e_i to S_w with a probability $p = k/n$
    else
      $w \leftarrow w + 1$ ▷ move the window by one step
      $j \leftarrow i - n$ ▷ j is the index of the expired element
      Add the items of S_{w-1} to S_w
      if $e_j \in S_{w-1}$ **then**
        Remove the item e_j from S_w
      end if
      Add e_i to the current sample S_w with a probability $p = k/n$
    end if
  end while
  return S_w
end procedure

We compare the performance of the SRS algorithm to that of the Chain+ sampling algorithm in terms of execution time. We consider several sampling rates k/n and we

choose a window size n equal to 10. The results are plotted in Figure 2.7. According to this figure, the Chain+ sampling algorithm has a slightly longer execution time than the SRS algorithm as an additional treatment is performed to avoid the redundancy in the sample and to guarantee exactly k distinct items in the sample of each window. Moreover, we notice that for both Chain+ sampling and SRS algorithms, the execution time grows for higher sampling rate, which can be explained by the additional time to store more items in the current sample.

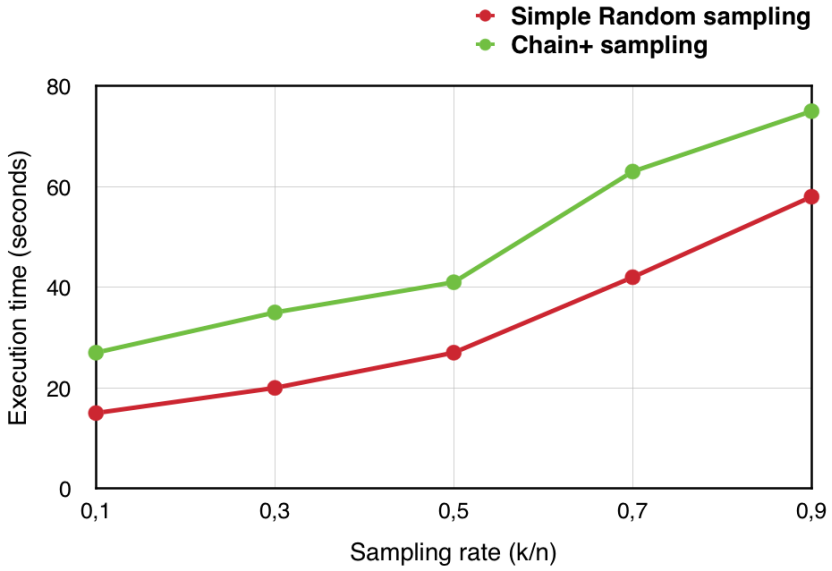


FIGURE 2.7 – Execution time of the Simple Random Sampling and Chain+ sampling algorithms, for different sampling rates k/n.

The principal advantage of the Chain+ sampling algorithm is that it provides an accurate sample (a sample with an exact size). In fact, the Chain+ algorithm provides at any time a sample of fixed size k among the n most recent elements of the stream, whereas the SRS algorithm gives a sample of variable size $\in [0, n]$. Figure 2.8 shows the experimental variation of the sample size when using the SRS algorithm. One can notice that this variation follows the theoretical binomial distribution $B(x, n)$ given by:

$$P(X = x) = \binom{n}{x} p^x (1-p)^{n-x}$$

where p is the sampling rate, k/n in our case.

Indeed, the size of the sample for a specific window can be seen as the number of successes (number of selected items to add to the sample) in n independent identical Bernoulli trials with probability p. As a conclusion, the Chain+ sampling algorithm is slightly slower than the SRS algorithm but it provides a more accurate sample. There is distinctly a trade-off between the quality of the sample and the cost of the used sampling algorithm expressed in terms of execution time.

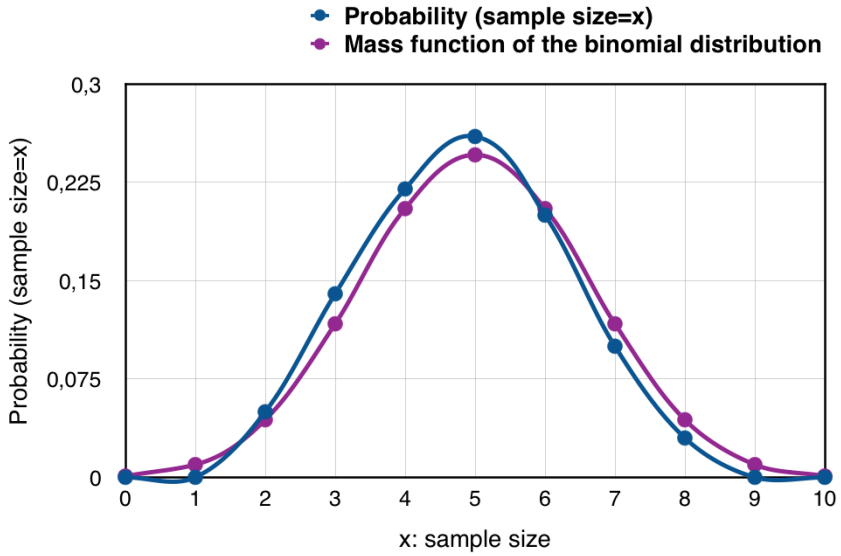


FIGURE 2.8 – Distribution of the sample size of the Simple Random Sampling algorithm without replacement.

2.4 Enhancing the Chain+ sampling algorithm

2.4.1 Inverting the selection for high sampling rates strategy

Our goal in this section is to improve the performance of the Chain+ sampling algorithm in terms of execution time. As we discussed earlier, the execution time of the Chain+ sampling algorithm increases when the sample size k is close to the size of the window n. This is due to the collision problem that gets worse when k/n is close to 1. According to Figure 2.5, when the sampling rate is less than or equal to 0.5, the size of the window slightly impacts the execution time of the algorithm. However, when k/n is greater than 0.5, there is a clear trade-off between the execution time and the window size n: the execution time becomes longer for a bigger window size. This represents the key idea of the "Inverting the selection for high sampling rates" strategy.

The main concept of this strategy is to reverse the role of the Chain+ sampling algorithm when the sampling rate exceeds 0.5. In such case, and as presented in Algorithm 4, the Chain+ sampling algorithm selects and removes the items that have to be excluded from the sample, and stores the remaining items. Since we are not interested in the contents or the values of the items to be removed, only the indexes of these items (and not the items themselves) are stored in the *chain-sample*. The other steps of the algorithm remain the same. Following the application of this strategy, and for a given sampling rate $p = k/n$, we estimate reducing the execution time of the Chain+ sampling algorithm to be approximately equal to that of $p' = 1 - p = k/n - 0.5$. Since $p > 0.5$, p' is smaller than p and the execution time of the Chain+ sampling with p' is lower than the execution time of Chain+ sampling with p, as we have less collisions.

Figure 2.9 illustrates an example of selecting $k = 6$ items from a window of size $n = 10$ using the "Inverting the selection for high sampling rates" strategy.

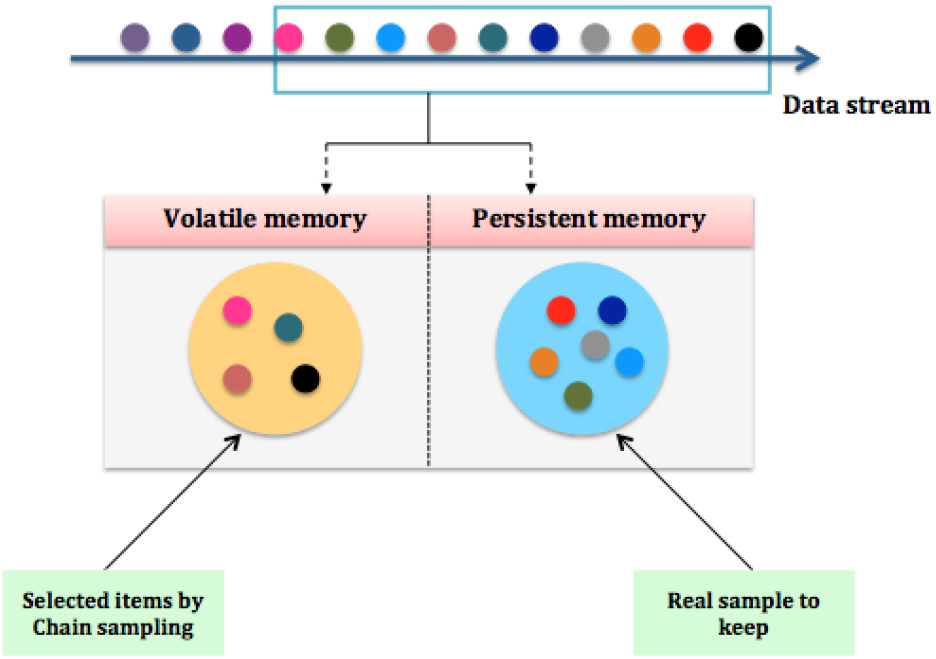


FIGURE 2.9 – Sampling $k = 6$ items over a sliding window of size $n = 10$ with the Chain+ sampling algorithm using the "Inverting the selection for high sampling rates" strategy.

Algorithm 4. Chain+ sampling algorithm using the "Inverting the selection for high sampling rates" strategy

procedure INVERTEDCHAIN+SAMPLING(k, n) ▷ k is the sample size, n is the window size
  if $k/n \leq 0.5$ **then**
    function CHAIN+SAMPLING(k, n)
      CHAIN+SAMPLING(k, n)
    end function
    return S
  end if
  if $k/n > 0.5$ **then**
    $k' \leftarrow n - k$
    function CHAIN+SAMPLING(k, n)
      CHAIN+SAMPLING(k', n)
    end function
    return $\overline{S}$
  end if
end procedure

2.4.2 Divide-to-Conquer strategy

As we discussed in Section 2.3.3, the execution time of the Chain+ sampling algorithm increases widely when the window size n is large and the sampling rate k/n is high. This is due to the collision problem. To overcome such situation, we propose in this section another improvement of the Chain+ sampling algorithm inspired by the "Divide-to-Conquer" strategy.

Let us consider the following principal query:

```
Select at random exactly k distinct items
among the n most recent items of the stream;
```

The key idea of the "Divide-to-Conquer" strategy is to divide the principal query into several sub-queries which will be performed on smaller adjacent windows. Indeed, we are susceptible to have fewer collisions with a sliding window of small size. Therefore, for a given constant $c \in [2, k]$, the principal query is divided into c sub-queries performed on c adjacent small windows each of size $n' = n/c$. Let i be the index of the most recent received item in the stream, the principal query selects randomly k items in the interval $[i - n + 1, i]$ such that the j^{th} sub-query selects randomly $k' = k/c$ distinct items in the interval:

$$[i - n + \frac{(j-1) \times n}{c+1} + 1, i - n + \frac{j \times n}{c}]$$

The initial window of size n is so split into c adjacent windows of each of size $n' = n/c$. Particularly, the c^{th} sub-query concerns the items $\in [i - n/c + 1, i]$ which depict the most recent small window.

Given the parameters of the principal query: window size n, sample size k, and splitting factor c, the Chain+ sampling algorithm maintains c independent samples, each of size $k' = k/c$. Each sample will be extracted from one of the c small adjacent windows of size $n' = n/c$ so that to form at the end $k = k' \times c$ samples. In other words, using the definition of *chain-sample* introduced in Section 2.3.2, k' *chain-samples* will be maintained and updated for each small window of size n'.

To answer the principal query at time i, the following instructions are executed:

- Consider the c adjacent small windows.
- For each small window and for each *chain-sample*, select the oldest sample belonging to this *chain-sample* with an index included in the considered small window.

Therefore, the principal query will be replaced by the following one:

```
Select at random exactly k/c distinct items
among the n/c most recent items of the stream;
```

To address the larger window issued from the principal query (k samples among the last n items of the stream); we propose to store, at any time, all the samples that did not expire (samples contained in the large window of size n), i.e. samples with indexes in $[i - n + 1, i]$. In addition, samples issued from each *chain-sample* must be stored separately.

Notice that each small window can contain more than one item belonging to a given *chain-sample*. This can be explained by the fact that with the Chain-sample algorithm, even with only one *chain-sample* ($k = 1$), the selected items can have successive indexes as the successor of a sampled item i is totally random in the interval $[i+1, i+n']$. Thus, a given small window can contain some samples belonging to the future sliding window. While executing each sub-query, we assume that the

considered small window is the current (most recent) sliding window. That is why we select the oldest sample from each small window. A given small window randomly chosen in the past contains certainly at least k' samples because it can be considered as a past sliding window. However, the last small window contains exactly k' samples as it is really the most recent sliding window.

An additional advantage of our proposed "Divide-to-Conquer" strategy is that it guarantees a uniform (but random) distribution of the samples in the big window of size n. This is achieved for each execution thanks to the introduction of the small windows, as it can be noticed in Figure 2.10. Using the standard Chain-sample algorithm, the uniform distribution is only achieved in average and is not guaranteed for each execution.

An example of sampling $k = 5$ elements from a window of size $n = 10$ using the "Divide-to-Conquer" strategy is presented in Figure 2.10. In this example, we choose the biggest splitting factor $c = k = 5$. Thus, the principal window of size $n = 10$ is divided into 5 sub-windows, each of size $k' = 2$.

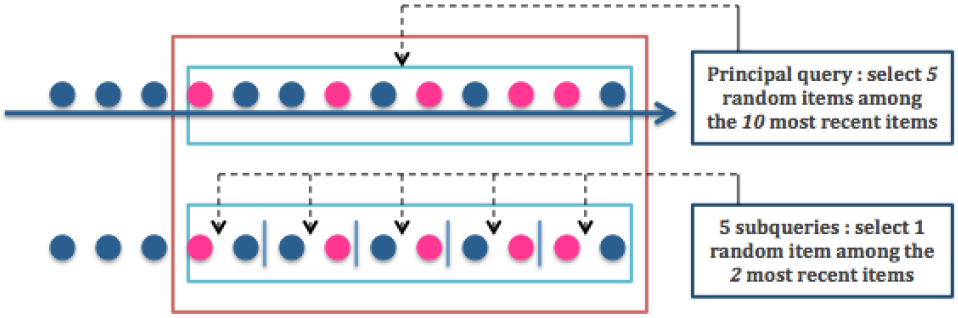


FIGURE 2.10 – Sampling $k = 5$ items over a sliding window of size $n = 10$ with the Chain+ sampling algorithm using the "Divide-to-Conquer" strategy.

2.4.3 Experimentations

Our goal in this section is to evaluate the efficiency of the two new enhanced versions of the Chain+ sampling algorithm: "Inverting the selection for high sampling rates" and "Divide-to-Conquer" strategies. The specifications of our machine are RAM: 4 GB, System Disk: 120 GB and Processor: 2.7 GHz Intel Core i5.

We study in Figure 2.11 the impact of the "Inverting the selection for high sampling rates" strategy on the execution time of the Chain+ sampling algorithm. We choose a window size $n = 1000$ items and we consider several sampling rates k/n. Recall that with the use of this strategy, the process of the Chain+ sampling algorithm is inverted only when the sampling rate is greater than 0.5. Therefore, the execution time with and without the application of this strategy remains the same when the sampling rate is ≤ 0.5.

When the sampling rate is > 0.5, the execution time of this strategy is sharply smaller than that of the Chain+ sampling algorithm. We can also notice that the execution time for a high sampling rate k/n is not the same as that for a sampling rate equal to $1 - k/n$. For instance, the execution time for $k/n = 0.8$ is higher than that for $k/n = 0.2$. This can be explained by the fact that for a sampling rate equal

to 0.2, only the selected items (20%) are stored. However, for a sampling rate equal to 0.8, not only the indexes of the removed items (20%) are stored, but also the kept items (80%) since they represent the targeted sample for the current window. This adds a small overhead in terms of execution time and memory consumption of the algorithm.

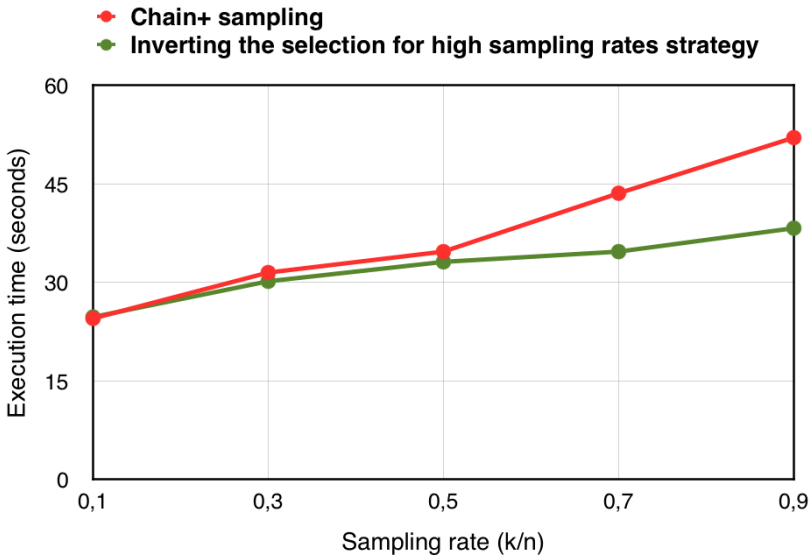


FIGURE 2.11 – Impact of the "Inverting the selection for high sampling rates" strategy on the execution time of the Chain+ sampling algorithm.

We evaluate in Table 2.1 the efficiency of the "Divide-to-Conquer" strategy by comparing the execution time of the Chain+ sampling algorithm with and without using this strategy. We choose a fixed sampling rate $k/n = 0.5$ and different window sizes n, and we use the highest splitting factor $c = k$. So, the Chain+ sampling algorithm is performed with the parameters $k' = 1$ and $n' = 2$ to randomly select half of the elements present in the last sliding window of size n. Two main conclusions can be inferred from this table. At first, one can notice that the execution time decreases distinctly when applying the "Divide-to-Conquer" strategy. For instance, for a window size equal to 1000 items, the execution time decreases to less than the half. Secondly, one can see that, even with the use of the "Divide-to-Conquer" strategy, the window size still has a significant impact on the execution time of the Chain+ sampling algorithm which increases with the increase of n. This can be explained by the fact that all the selected items in the last sliding window are stored in the memory.

2.5 Conclusion

In this chapter, we studied in depth the Chain-sample algorithm. We identified a particular weakness of this algorithm caused by the collisions problem. The collision occurs when the same element is selected to be included in the current sample more than once during the execution of the algorithm. This problem leads to the degradation of the quality of the samples over time and worsens with the increase of the sampling rate k/n. In order to overcome this problem, we presented a new

TABLE 2.1 – Execution time reduction of Chain+ sampling algorithm using the "Divide-to-Conquer" strategy, for a sampling rate $k/n = 0.5$, a splitting factor $c = k$, and for different window sizes n

Chain+ version / k, n	without the "Divide-to-Conquer" strategy	with the "Divide-to-Conquer" strategy
$k = 5$, $n = 10$	$19,81\ sec$	$18,79\ sec$
$k = 250$, $n = 500$	$33,43\ sec$	$19,96\ sec$
$k = 500$, $n = 1000$	$52,85\ sec$	$21,63\ sec$
$k = 1500$, $n = 3000$	$60,15\ sec$	$40,96\ sec$

version of the algorithm, called Chain+, which is intended to ensure that the sample on each window contains exactly k distinct items. Because of this alteration, the execution time of the Chain+ sampling algorithm has increased amply, especially, when the sample size k is close to the window size n.

To overcome this problem and to reduce the execution time of the algorithm, we proposed two strategies. The first one is called "Inverting the selection for high sampling rates" strategy, and the second approach is inspired by the "Divide-to-Conquer" strategy. With the "Inverting the selection for high sampling rates" strategy, the role of the Chain-sample algorithm is reversed: it identifies the elements that should not be added to the sample. Thus, all the items that do not belong to the sample will be stored. The results of the experiments clearly showed that the execution time needed to sample the data is considerably reduced. The principal idea of the second approach called "Divide-to-Conquer" is to divide the main window into several small and adjacent sub-windows and to execute the Chain+ sampling algorithm with the parameters $k' = k/c$ and $n' = n/c$ such that c is a constant $\in [2, k]$. The experiments showed that the execution time becomes much shorter using this strategy. In addition to reducing the execution time, this "Divide-to-Conquer" strategy ensures a uniform and random sample distribution in the large window.

Other experiments were carried out to study the size of the *chain-sample* (the number of elements and indexes of successors stored in memory) according to the window size n. The obtained results showed that there is no direct relationship between n and the average length of the *chain-sample*, which confirms the theoretical result. Our experiments also showed that the experimental length of the *chain-sample* exceeds sometimes the theoretical upper bound $e = 2.72$, but is almost always lower than the high probability upper bound $log(n)$.

The size of the sample and the sampling algorithm have a noticeable effect on the accuracy of the estimation drawn from the sampled data, so that they can be specified to achieve a certain level of accuracy while respecting a bounded memory and a reasonable execution time. For instance, Provost *et al.* [Provost et al., 1999] presented a progressive sampling algorithm based on the SRS algorithm. Their key idea is to build a sample of small size and increase it gradually until the accuracy of the sample is no longer improved. The choice of the sampling algorithm and sampling rate, and their impact on the accuracy of the statistical estimations will be discussed in detail in the next chapter.

Chapter

3 On the impact of data sampling on data streams statistics inference

Contents

3.1 Introduction **50**

3.2 Sampling impact on the queries estimation accuracy **50**

3.2.1 Chain+ sampling algorithm 51

3.2.2 A bounded-space SRS algorithm without replacement over a purely sliding window 51

3.2.3 Deterministic sampling algorithm 52

3.2.4 Experimentations 53

3.3 Sampling impact on the anomalies detection **57**

3.3.1 Problem definition 57

3.3.2 EWMA control chart algorithm 59

3.3.3 A bounded-space SRS algorithm without replacement over a jumping sliding window 59

3.3.4 A bounded-space WRS algorithm without replacement over a jumping sliding window 60

3.3.5 Experimentations 61

3.4 Conclusion **65**

The scientific contribution and the obtained results presented in the second section of this chapter have been published in our paper: **Rayane El Sibai**, Yousra Chabchoub, Raja Chiky, Jacques Demerjian and Kablan Barbar. *"A performance evaluation of data streams sampling algorithms over a sliding window"*. In the IEEE Middle East & North Africa COMMunications Conference, IEEE, 2018.

3.1 Introduction

As we have seen in the previous chapters, data streams sampling is intended to build a sample on which the future data analysis tasks will be performed. The effectiveness of the sample depends on several parameters: the used sampling algorithm, the chosen sampling rate k/n, and the window size n if the sliding window model is adopted.

Our goal in this chapter is to answer the following question:

Problem 1. *Given a stream of items, what is the most relevant sampling technique to use in our context? What are the right parameters k and n to choose?*

To answer this question, we will study in this chapter the impact of sampling in terms of (1) the similarity between the aggregation queries results on the original data and the sampled data, and (2) the result of anomalies detection of the sampled data, measured in terms of the true positive rate called *recall*.

Several studies have been dedicated to examine the impact of data sampling. Mai *et al.* [Mai et al., 2006] presented an in-depth study of the effect of data sampling on anomalies detection of traffic measurements coming from high-speed IP-backbone networks. Several sampling methods were applied to sample the IP packet traces. The authors assessed the impact of sampling on port scans detection, volume anomaly detection, and data characteristics, namely, the variance. The impact of IP packets sampling was also addressed in [Brauckhoff et al., 2006, Pescapé et al., 2010, Zhang et al., 2016]. In [Xu et al., 2015], the authors studied the impact of sampling on the analysis of tweets generated on social networks. This analysis includes the study of tweets volume, tweets distribution, and user influence measured by his comments and retweet activities.

This chapter is organized as follows. We discuss in Section 3.2 the sampling accuracy defined as the accuracy of the aggregation queries performed on sampled data. Three data streams sampling algorithms are considered: Chain-sample, Deterministic sampling and Simple Random Sampling (SRS). At first, we adapt the Deterministic sampling and SRS algorithms to the sliding window concept by maintaining a sample of the most recent elements of the stream. After that, we compare the performance of the three algorithms by studying the impact of sampling rate and sliding window size on sampling accuracy and execution time. We study in Section 3.3 the impact of data sampling on the anomalies detection results using Exponentially Weighted Moving Average (EWMA) control chart algorithm. We end the chapter with a conclusion.

3.2 Sampling impact on the queries estimation accuracy

When building a data stream sample, many issues are encountered if the sliding window model is adopted. They concern particularly:

- The preservation of a sample of fixed size over each window.

- The replacement methodology of the expired items in the sample.

Therefore, our problem can be defined as follows:

Problem 2. *Given a data stream, how to maintain, at every time, a sample of fixed size k items over a tuple-based sliding window of size n?*

To answer this question, we provide in the following a study of different approaches based on three sampling algorithms: Simple Random Sampling (SRS), Deterministic sampling and Chain+ Sampling.

3.2.1 Chain+ sampling algorithm

We recall that the main purpose of the Chain-sample algorithm [Babcock et al., 2002] is to provide, at any time, a sample of fixed size k from the most recent n elements of the stream. Assuming that the stream is composed of elements with an ever-increasing index, the Chain-sample algorithm maintains k independent samples over each tuple-based sliding window of the stream in order to provide a sample of size k. Each sample is constructed as follows: In the first window of the stream, each item is selected with a probability equals $\frac{min(i,n)}{n}$, as soon as it arrives, where i is the index of the item in the window and n is the window size. Once chosen, a successor's index j for the i^{th} item is chosen randomly among the items with indexes $\in [i+1, i+n]$. When the item with index j arrives at the window, it will be saved in the memory and a random successor for it will be chosen with the same method as for the item i. When the item with index i expires, it will be removed from the sample and replaced by its successor j. In its basic version, the Chain-sample algorithm suffers from a major issue: the redundancy in the sample caused by the collision problem occurring when the sample size k is greater than 1. We introduced in Section 2.3 a new version of this algorithm, called Chain+ sampling algorithm, designed to deal with the collision and redundancy problems. An enhanced version of the Chain+ version, called "Inverting the selection for high sampling rates", was also proposed in Section 2.4.1. Its goal is to enhance the execution time of the Chain+ version. It is this strategy of the Chain+ sampling algorithm that will be used in the incoming experiments of this section.

3.2.2 A bounded-space SRS algorithm without replacement over a purely sliding window

We recall that the Simple Random Sampling (SRS) is a probabilistic method and the simplest way to build a random sample. With the SRS algorithm, each element of the original data has the same chance of being sampled. Nevertheless, in its basic version, the SRS algorithm does not use the data streams sliding window model, and if adapted to this context as presented in Section 2.3.4, it gives a sample of a random size $k \in [0, n]$ where n the window size. The SRS without replacement algorithm can be easily adapted to our context. It will be executed as follows to build and to maintain a sample of fixed size k on a tuple-based sliding window of size n of the stream: In the first window of size n of the stream, SRS selects each item with a

probability equals $p = k/n$. This step will be repeated until exactly k distinct items are selected. In the upcoming windows, when a sample item i expires (becomes outside the current window), it will be excluded from the sample, and a random replacement r for it will be chosen among the items $\in [i+1, i+n]$ by performing a random sampling with a probability equals $p = k/n$ on the items of the current window, while beginning with the most recent item. This step will be repeated until a replacement item is selected. The latter will be added to the sample.

The detailed pseudo code of the SRS algorithm without replacement over a purely sliding window is given by Algorithm 5.

Algorithm 5. A bounded space Simple Random Sampling without replacement over a purely sliding window

```
1: procedure SRSPurelyWindow(k, n) ▷ k is the sample size, n is the window size
2: Initialization: S_w is the sample for the window w, at the beginning S_1 = ∅
3:     repeat
4:         for i ∈ [1, n] do
5:             Add e_i to S_w with a probability p = k/n
6:         end for
7:     until selecting k distinct items in S_w
8:     while a new item e_i is received, (i > n) do
9:         w ← w + 1                                   ▷ move the window by one step
10:        j ← i − n                                   ▷ j is the index of the expired element
11:        Add the items of S_{w−1} to S_w
12:        if e_j ∈ S_{w−1} then
13:            Remove the item e_j from S_w
14:            repeat
15:                for each item e in the current window w, starting with the most
   recent item do
16:                    if the size of S_w is < k then
17:                        Add e_i to S_w with a probability p = k/n
18:                    end if
19:                end for
20:            until selecting an item that does not exist in S_w
21:        end if
22:    end while
23:    return S_w
24: end procedure
```

3.2.3 Deterministic sampling algorithm

The easiest manner to construct a data stream sample is to use the Deterministic sampling algorithm. It is is a non-probabilistic technique with which the sample is built without randomness. It consists of selecting one item every $1/p$ items of the stream. We assume that the stream consists of items with an always-increasing index. To meet our need of having, at any time, a sample of exactly k distinct items among the n most recent items of the stream, the Deterministic sampling algorithm is designed as follows:

1. Select each incoming item of the stream if its index equals $x \times n/k$ where x is a positive integer.

2. When an item i expires (becomes outside the current window), it will be removed from the sample if it is present in it.

Algorithm 6. Deterministic sampling over a sliding window

procedure DETERMINISTICSAMPLING(k, n) ▷ k is the sample size, n is the window size
Initialization: S_w is the sample for the window w, at the beginning $S_1 = \varnothing$
  while a new item e_i is received **do**
    $i \leftarrow i + 1$ ▷ i is the index of e
    $w \leftarrow w + 1$ ▷ move the window by one step
    if $i = x \times n/k$ where $x > 0$ **then**
      Add e_i to S_w
    end if
    $j \leftarrow i - n$ ▷ j is index of the expired element
    Add the items of S_{w-1} to S_w
    if $e_j \in S_{w-1}$ **then**
      Remove e_j from S_w
    end if
  end while
  return S_w
end procedure

3.2.4 Experimentations

This section is intended to compare the performance of the SRS, Deterministic sampling and Chain-sample algorithms in terms of execution time and sampling accuracy. We recall that for the Chain+ sampling algorithm, we will use the "Inverting the selection for high sampling rates" strategy proposed in Chapter 2.

3.2.4.1 Dataset

We use the dataset presented in the introduction of the report (WAVES dataset). We recall that dataset consists of data streams issued from flowmeters delivering water to a big French city. The duration of each stream is of 21 months, from January 2013 to September 2014. The data are periodically generated by the sensors with a frequency of one observation each 15 minutes. Each recorded observation comprises two fields: the timestamp designating the recording time of the measure, and the attribute value representing the delivered water volume in m^3. The volume of the water consumed by a given sector is an algebraic sum of the flows delivered by its associated flowmeters. All the experiments in this section will be performed on the summed data rather than the flowmeters data themselves. We recall that the specifications of our machine are RAM: 4 GB, System Disk: 120 GB and Processor: 2.7 GHz Intel Core i5.

3.2.4.2 Execution time

We evaluate in Figure 3.1 the computational resources of the three algorithms in terms of execution time needed to summarize a given stream, for different sampling rates. The size of the window n is fixed to 10. Regarding the Chain-sample algorithm, we use the "Inverting the selection for high sampling rates" version. It is the improved version of the Chain+ algorithm that reduces the execution time for high sampling rates. The results show that the Deterministic sampling algorithm has the smallest execution time compared to SRS and Chain algorithms, as it is the simplest sampling algorithm. Regarding SRS, the redundancy in the sample occurs when the sampling rate k/n is > 0.1. This happens when the same item is selected many times to be added to the sample on the same sliding window. This problem becomes more severe when k is close to the size of the window n. To avoid this duplication, and to provide exactly k distinct items in each sample, the selection procedure must be repeated until selecting an item that is not already present in the current sample. This condition adds a considerable overhead in terms of execution time, especially when the sampling rate is high (k is close to n). The difference in the execution time for SRS and Chain algorithms increases with the increase of the sampling rate k/n. This difference becomes clearer when k/n is greater than 0.5. This is due to the use of the "Inverting the selection for high sampling rates" strategy which reduces the collision rate to be equal to that of a sampling rate of $1 - k/n$ when k/n exceeds 0.5, and thus, reduces the execution time of the Chain-sample algorithm.

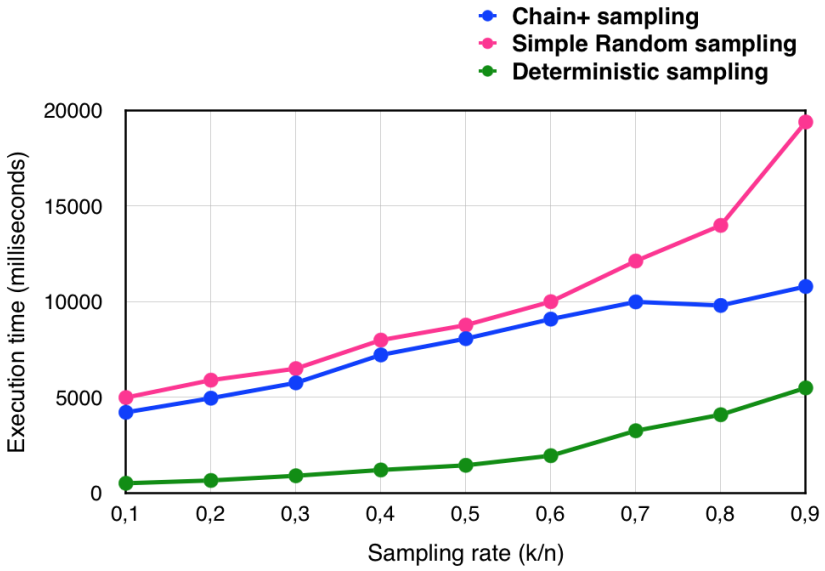


FIGURE 3.1 – Execution time of the Deterministic, Simple Random and Chain+ sampling algorithms over a purely sliding window, for different sampling rates k/n.

3.2.4.3 Sampling accuracy

When evaluating a query on the most recent data of the stream, we only have a set of samples built on the windows of this stream. Thus, the response to the query can be provided by estimation. We study in this section the impact of the sampling on the quality of the estimation and we limit the analysis to the MEAN aggregation

query. Our choice of this query is motivated by the fact that it is a statistic that allows us to measure the average water consumption of users over a certain period of time, which will allow us to study and analyze the users' behavior and its evolution over time.

Let m be the true mean calculated from the original data in a given window. The empirical mean $\overline{X}$ of the window sample is an unbiased estimator of m, calculated as follows:

$$\overline{X} = \frac{1}{k} \sum_{i=1}^{k} e_i$$

where k is the size of the sample, and e_i is the value of the item of index i in the sample. The accuracy of the mean estimation is quantified by the relative error given by:

$$error = \left| \frac{m - \overline{X}}{m} \right| \times 100\%$$

There are two parameters that affect the sampling accuracy: the sampling rate k/n and the window size n. Actually, we can achieve a sampling rate p while sampling with different sample sizes k and window sizes n, such that:

$$p = \frac{k}{n} = \frac{x \times k'}{x \times n'}$$

where x is a positive integer.

We study in Figure 3.2 the impact of the sampling rate k/n and the window size n on the mean estimation error of the Chain+ sampling algorithm. One can notice that the window size n has a notable impact on the mean estimation error. Indeed, for a given sampling rate k/n, the error decreases with the increase of n. Also, for a given window size n, the error decreases with the increase of k/n. These results can be explained by the law of large numbers which states that the sample mean tends towards the true mean as the number of trials (sample size in our case) increases [Gravetter and Forzano, 2018] which improves the accuracy and representativity of the sample.

We study in Figure 3.3 the impact of the collision problem of Chain-sample algorithm on the accuracy of the estimation. We compare the mean estimation error for the two versions of the Chain-sample algorithm: the traditional version and Chain+ version. For that, we compute the mean of the sample constructed on each window of the stream, and we compare it to the real value of the mean on this period of time. The results show that the estimation error with the traditional Chain algorithm is much higher than that with Chain+. This is due to the redundancy problem causing the degradation of the quality of the samples.

According to Figure 0.3, we can notice a periodicity in the water consumption, related to the human activity. We study in Figure 3.4 the impact of the data periodicity on the accuracy of the estimation. The data are sampled using the Chain-sample, SRS and Deterministic sampling algorithms with a very small sampling rate. In

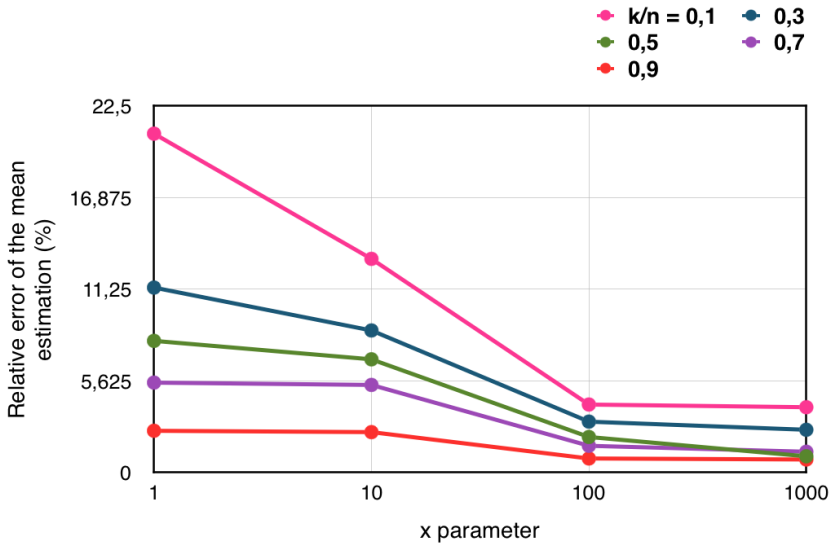


FIGURE 3.2 – Impact of the sampling rate and window size on the mean estimation error of the Chain+ sampling algorithm, for different sampling rates k/n

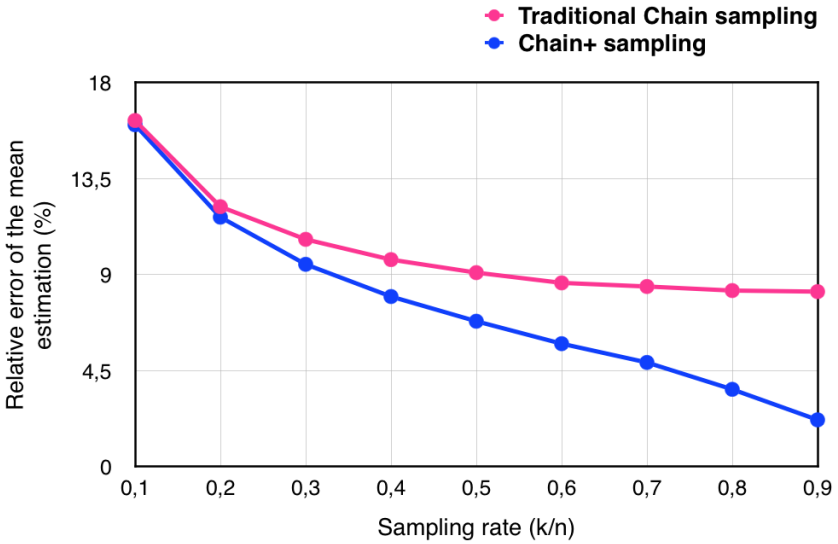


FIGURE 3.3 – Impact of the collision problem on the mean estimation error of the Chain-sample algorithm, for different sampling rates k/n.

fact, in degraded environments such Wireless Sensor Networks (WSN), the sensors are devices with limited resources in terms of battery power, CPU, memory, and bandwidth [Jain and Chang, 2004]. Because of the memory limitations, WSN devices cannot store a lot of information. However, each node must compute, process and transmit its data to other nodes or to the central system. Implementing the sampling process on WSN devices and using a low sampling rate can be a solution to deal with the memory and time constraints.

Figure 3.4 depicts the sampling accuracy for SRS, Chain and Deterministic sampling algorithms in such degraded environments. As shown in this figure, the Deterministic sampling algorithm has the highest estimation error, and SRS and

Chain-sample algorithms give both similar results. This can be explained by the fact that when the period of the data is smaller than the sampling rate, the sampled data with the Deterministic sampling algorithm will be unrepresentative of the window as a whole. In our case, the selected data often belong to a peak of water consumption, leading to a high error of the mean estimation. On the contrary, the Chain-sample and the SRS algorithms are more representative because the elements are selected randomly in an independent manner of the data periodicity. One can notice that the size of the window has an impact on the sampling accuracy of the three sampling algorithms. The decrease of the error for a high window size can be explained by the law of large numbers.

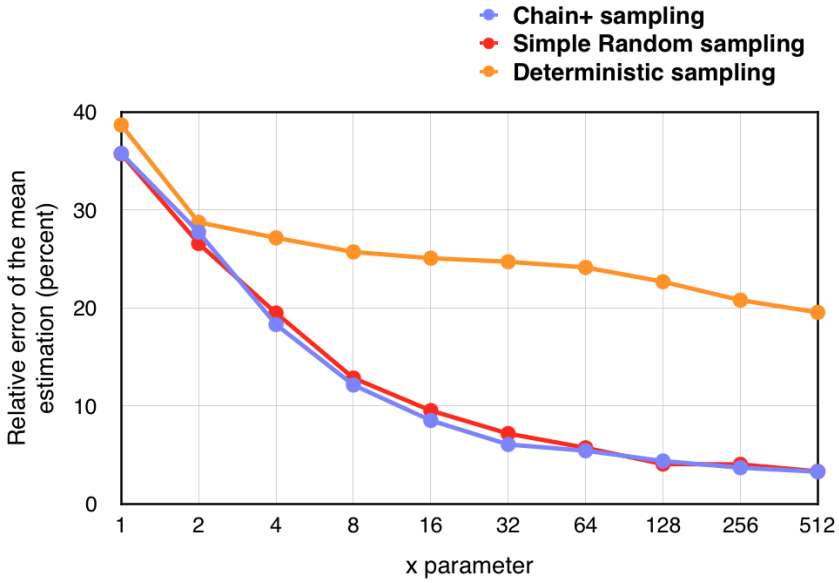


FIGURE 3.4 – Mean estimation error of the Deterministic, Simple Random and Chain+ sampling algorithms for a sampling rate of 1 observation per 12 hours, for different window sizes.

3.3 Sampling impact on the anomalies detection

3.3.1 Problem definition

Anomalies detection of the water consumption is a primordial step to supervise the water distribution network since it is a key parameter for the detection of water leaks. Indeed, following this step, the analysis of the detected anomalies will be performed by the monitoring system, which helps the network explorer to detect the potential leaks and make the right decisions. These anomalies are manifested by a large increase in the water consumption volume during a certain period of time. Yet, because of the sampling process, many data related to the water consumption are lost. In this section, we study the sampling impact on the anomalies detection phase when this later is performed on the data stream summary instead of the original data. Note that the analysis of these anomalies and the determination of their potential causes (leaks, special events, etc.) is out of the scope of this thesis.

Since the water consumption volume of a given sector is given by the algebraic sum of the flows delivered by its associated flowmeters, the anomalies detection of a particular sector has to be performed on the summed data rather than the flowmeters data. To evaluate the sampling impact on the anomalies detection results, two cases are considered, as shown in Figure 3.5:

- Without sampling: the sum of the flowmeters of a given sector is performed, and the anomalies are therefore detected.
- With sampling: the flow of each flowmeter is sampled. Thereafter, the sum of the water consumption of the sector is calculated as the algebraic sum of the sampled flows of its associated flowmeters. Finally, the anomalies are detected.

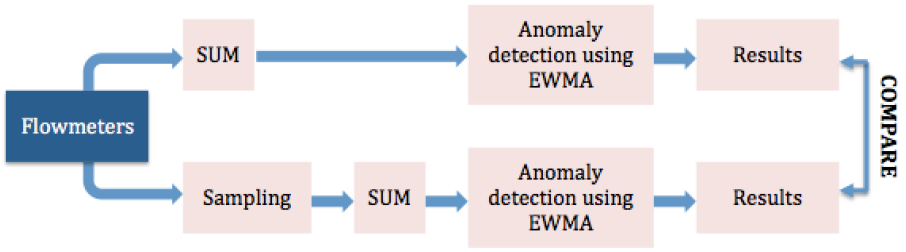


FIGURE 3.5 – Experiments' strategy.

We choose to sample the flow of each flowmeter using four sampling algorithms: Deterministic sampling, SRS, Chain-sample and Weighted Random Sampling (WRS).

As our ultimate goal is to detect the anomalies in the water distribution network manifested by very high water consumption, it is necessary to ensure that the data involving important delivered water volume are sampled. Missing such information due to the sampling process will result in a false response to the queries, and an important information to the network explorer will be lost. Therefore, it is indispensable to force the sampling of these data by giving them higher chance to be sampled. This is accomplished by the use of a weighted sampling algorithm. Notice that the use of such algorithm was not considered in the previous section since it is not appropriate for the mean estimation of the data. Indeed, because of the weighting, some data will contribute more than others to the mean estimation making this latter biased.

In contrary to the previous section, the jumping sliding window model will be used in this section. Recall that with the jumping window model the offset between two successive windows is equal to the window size. This choice is motivated by the fact that the sum of the flowmeters' has to be performed at different moments and each value from the data stream has to be included only once in the sum. Jumping windows are therefore well adapted to this context.

Several algorithms can be used to detect the anomalies. In this section, we choose to use the EWMA control chart algorithm since it is well adapted to our objective of detecting shifts of the process mean. Note that the anomalies detection algorithms will be discussed in more details in Chapter 5.

3.3.2 EWMA control chart algorithm

The EWMA control chart proposed by [Roberts, 1959] is designed for monitoring a process over time and detecting a change in the process from the in-control state to the out-of-control state. To monitor the process, EWMA maintains a weighted moving average (EWMA statistic) which gives a high priority for recent items. The weights for the observations are periodically updated and become smaller as the observations become older. EWMA control chart is widely used for anomalies and intrusion detection in the network [Bošnjak and Cisar, 2010, Čisar and Čisar, 2011]. EWMA statistic at time t is computed as follows:

$$ewma_t = \lambda \times e_t + (1 - \lambda) \times ewma_{t-1}$$

where:

- e_t is the observation at time t.
- $ewma_t$ is the EWMA statistic at time t.
- λ is the weighting factor $\in]0, 1]$: The choice of λ determines the impact of the previous value of the moving average on the calculation of the current EWMA statistic. For $\lambda = 1$, only the current observation (e_t)is involved in the calculation. When λ is small, the previous value of the moving average has more influence on the calculation than the current observation.

EWMA control chart reports an out-of-control alarm at time t when EWMA statistic exceeds the Upper Control Limit (UCL) or is lower than the Lower Control Limit (LCL). UCL and LCL are computed as follows:

$$UCL = ewma_0 + l \times \sigma_0 \times \sqrt{\frac{\lambda}{2 - \lambda}}$$

$$LCL = ewma_0 - l \times \sigma_0 \times \sqrt{\frac{\lambda}{2 - \lambda}}$$

where:

- $ewma_0$ is the mean of the training data, and represents the target mean value.
- σ_0 is the standard deviation of the training data.
- l is a positive coefficient.

$ewma_0$ and σ_0 are calculated from the training data. For this purpose, a learning window containing a small proportion of the data to be monitored is used to calculate $ewma_0$ and σ_0.

3.3.3 A bounded-space SRS algorithm without replacement over a jumping sliding window

The construction of a sample on a jumping sliding window using the SRS algorithm returns to building a new sample on each window of the stream while discarding the sample built on the previous window. The SRS algorithm can be adapted as follows

to construct a sample of fixed size k on a jumping sliding window of size n: In each window of the stream, the sample of the previous window is firstly discarded. Then, in the current window, select each item with a probability equals $p = k/n$. This step will be repeated until selecting exactly k distinct items. These steps are detailed in Algorithm 7.

Algorithm 7. A bounded space Simple Random Sampling without replacement over a jumping sliding window

procedure SRSJumpingWindow(k, n) ▷ k is the sample size, n is the window size
Initialization: S_w is the sample for the window w, at the beginning $S_1 = \varnothing$
  while a new item e_i is received **do**
    $i \leftarrow i + 1$ ▷ i is the index of e
    if $i = x \times n$ **then** ▷ when the w^{th} window is filled
      $w \leftarrow w + 1$ ▷ move the window by one step
      $S_w = \varnothing$
      repeat
        for $i \in [i - n + 1, n]$ **do** ▷ for each item e in the current window w
          Add e_i to S_w with a probability $p = k/n$
        end for
      until selecting k distinct items in S_w
    end if
  end while
  return S_w
end procedure

3.3.4 A bounded-space WRS algorithm without replacement over a jumping sliding window

The aim of the Weighted Random Sampling (WRS) algorithm is to construct a sample in which the inclusion probability of each item is determined by its weight with respect to the weights of the other items in the current window. There are several ways to define the weights, as presented in Section 1.5. The main concern with these methods is that they all generate the weights in a random manner. Nevertheless, one may be interested in sampling some items of the stream according to their values, timestamp, or other characteristics. Since our objective in this section is to detect the anomalies in the water distribution network manifested by very high water consumption, it is necessary to ensure that the data involving important delivered water volume are sampled. Therefore, it is indispensable to force the sampling of these data. This is done by giving them greater weights compared to their neighbors in the current tuple-based sliding window. The ability to sample the data according to their weights can be provided by the WRS algorithm with which the inclusion probability of each item can be proportional to its value. Thus, the smaller the value of the item, the lower is its probability of being sampled.

We propose in this section a WRS algorithm without replacement to construct a data stream sample over a jumping window. The expected performances of such an algorithm are:

- Sample each item according to its value with respect to the other items values in the current window.
- Provide a uniform sample of fixed size.

Thus, we propose to sample each item e_i in the window w with a probability equal to:

$$p_{e_i} = \frac{e_i}{max_n(e)}$$

where $max_n(e)$ is the maximal value in the window of size n.

In order to guarantee a uniform distribution of the sampled items in each window, we are inspired by the "Divide-to-Conquer" strategy presented in Section 2.4.2. Note that with the probabilistic algorithms (SRS and WRS), the sampled data of the flowmeters of a given sector do not have certainly the same timestamps. Thanks to the uniformity of the sample, the sampled data will have closer indexes, which avoids the addition of data having very distant timestamps. Recall that the Divide-to-Conquer strategy consists of splitting the main query into sub-queries made on smaller adjacent windows. Therefore, given the initial parameters window size n and sample size k, the WRS algorithm will be performed with the parameters $n' = n/k$ and $k' = 1$. Thus, each sample will be extracted from one of the k small adjacent windows of size $n' = n/k$ so that to form at the end k samples.

To construct a sample of exactly k items from each jumping window, the WRS algorithm will be executed as follows:

- Split the initial window of size n into k adjacent windows of size $n' = n/k$.
- For each small window select randomly $k' = 1$ item e_i included in the considered small window, with a probability equal to $p = \frac{e_i}{max_{n'}(e)}$, where $max'_n(e)$ is the maximal value in the small window.

Figure 3.6 shows an example of the flowmeters sum when sampling $k = 5$ items over a jumping sliding window of size $n = 10$ using the WRS algorithm. For each flowmeter, the jumping sliding window is split into $k = 5$ successive small window each of size $n' = 2$ and $k' = 1$ item is then selected from each small window of size $n' = 2$.

3.3.5 Experimentations

Our objective in this section is to detect the water consumption volume anomalies of a sector. Recall that the volume of the water consumed by a given sector can be simply inferred in real-time as an algebraic sum of the flows delivered by its associated flowmeters. We use the dataset presented in the introduction of the report (WAVES dataset) and also used in the previous section (Section 3.2.4.1). The chosen sector consists of 4 flowmeters. As the considered anomalies engender an increase of the mean of the data, we only focus on the control limit UCL. Two scenarios are considered and compared in terms of the true positives rate, *recall*, calculated as follows:

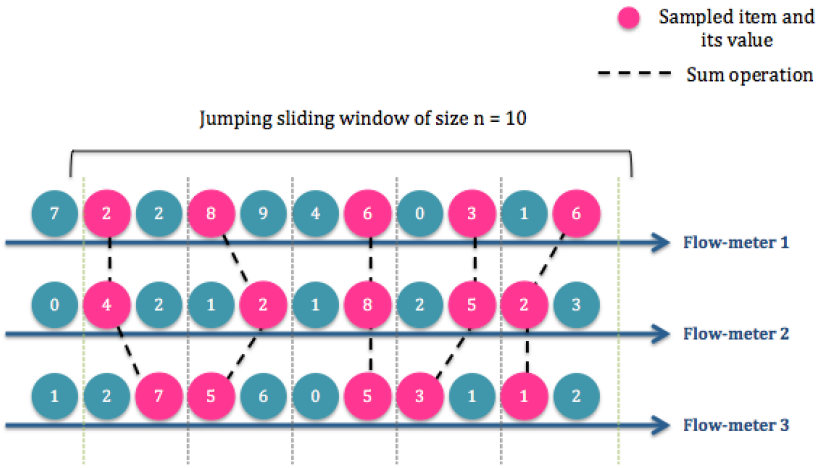


FIGURE 3.6 – Sum of the sampled flowmeters of a sector when sampling $k = 5$ items over a jumping sliding window of size $n = 10$ using WRS algorithm.

$$Recall = \frac{TP}{TP + FN} = \frac{Number\ of\ detected\ anomalies\ points}{Total\ number\ of\ anomalies\ points}$$

The increase of the water consumption of a given sector can be manifested in two ways: (1) the water consumption increases significantly at the scale of a single flowmeter. This increase is visible in both the data of the flowmeter in question and in the water consumption sum of the sector and (2) the water consumption of all the flowmeters of the sector increases slightly. These two scenarios are considered in the experiments.

For the first scenario, to inject the anomalies in the data, we chose one flowmeter and we inject in it at 3 random moments 3 anomalies. At a random instant t and we replaced e_t by $\mu_0 + 5\sigma_0$, for 20 successive values corresponding to 5 hours beginning from the instant t. μ_0 and σ_0 are the mean and standard deviation of the flowmeter data calculated during the training phase of EWMA. We repeated this mechanism 3 times to inject 3 errors. We obtained a total number of 60 injected anomalies points. For the second scenario, for each one of the 4 flowmeters of the sector, we inject at the same moments t considered in the first scenario 3 different anomalies. For each instant t, we replaced e_t by $(\mu_0 + 5\sigma_0)/4$, for 20 successive values beginning from the instant t. We obtain the same total number of 60 injected anomalies points. The data with the injected anomalies are plotted in Figure 3.7.

EWMA statistic at each time t, EWMA(t), of the original data (without sampling) are plotted in Figure 3.8. We use the first seven days of the dataset to compute the parameters of EWMA control chart: the target mean value $ewma_0$, the standard deviation σ_0, and the Upper Control Limit (UCL).

The obtained results for the first scenario are plotted in Figure 3.9. Several conclusions can be drawn from this figure. At first, one can notice that even without sampling, many injected anomalies points are missed since the recall value is equal to 56.66. In fact, according to [Lucas and Saccucci, 1990], with the used parameters ($\lambda = 0.25$ and $L = 3$), the Average Run Length (ARL) needed by EWMA algorithm to detect a process mean shift of σ is equals 11 successive anomalies points that exceed $\mu_0 + \sigma_0$. Since EWMA is applied to the sector's summed data and not to the

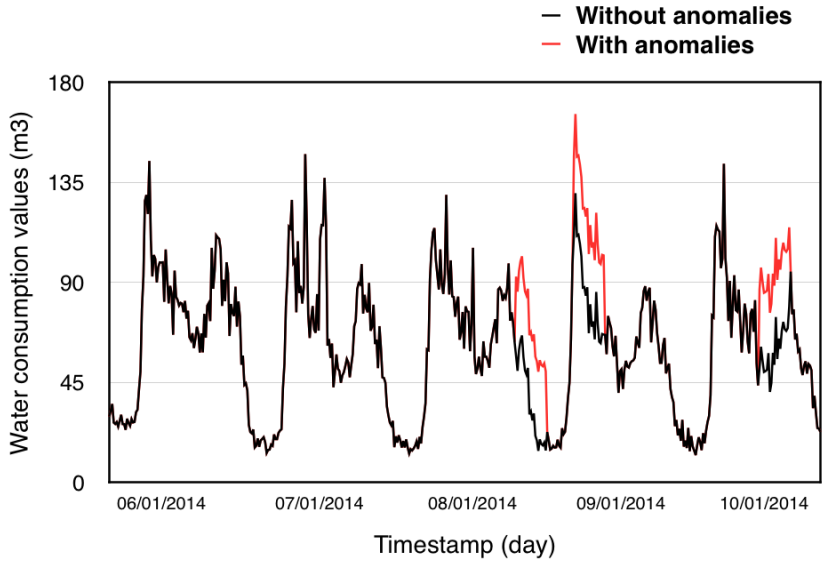


FIGURE 3.7 – Volume of the water consumed by the sector, over time, with and without anomalies.

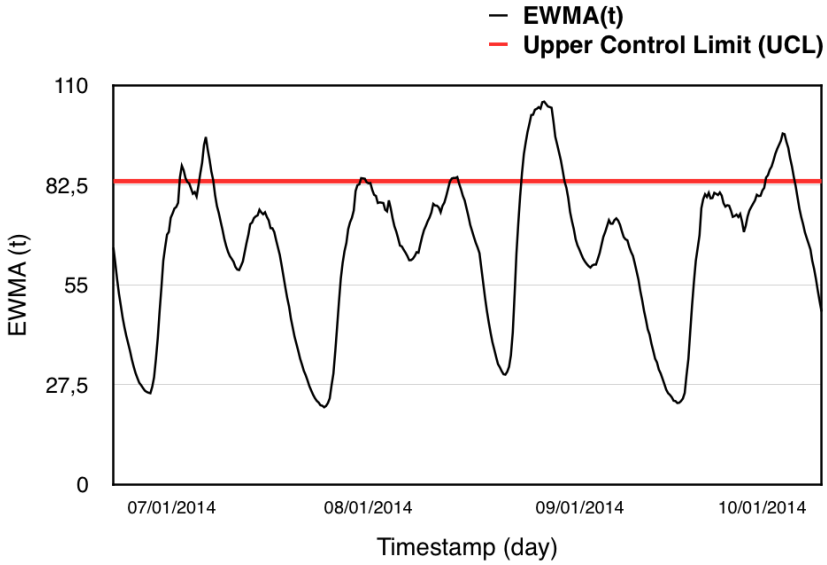


FIGURE 3.8 – EWMA control chart.

flowmeters data themselves, the magnitude of a shift injected in the flowmeter data decreases after the sum and may not exceed $\mu_0 + \sigma_0$.

Moreover, we note that for a very small sampling rate, no sampling method could detect the anomalies in the data. Also, we see that whatever the sampling rate, the WRS algorithm has a higher performance compared to the other algorithms. Moreover, as the sampling rate increases, the performance of the three algorithms increases to be close to the results obtained without sampling. Finally, we can notice that the Chain-sample algorithm gives very similar results to the SRS algorithm in terms of detected anomalies as these algorithms are both probabilistic and provide random samples.

The obtained results of the first and second scenarios are depicted by Figure 3.10.

One can notice that the performances of the SRS and WRS algorithms vary slightly between the two scenarios. Notice that with the Deterministic sampling algorithm, the obtained results are the same regardless of the considered scenario. This is explained by the fact that for a specific sampling rate k/n and a specific window size n, the same data are always sampled whatever the scenario.

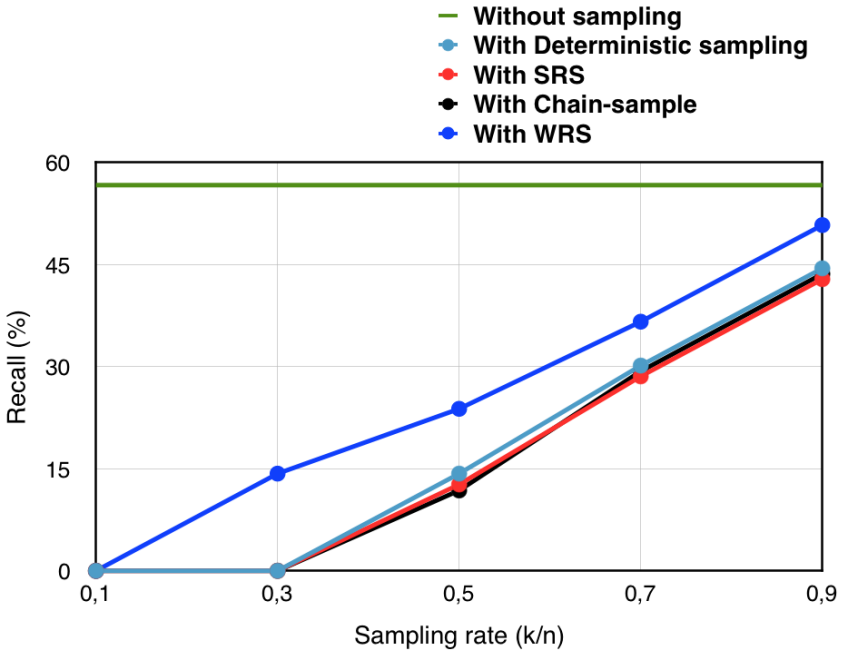


FIGURE 3.9 – Impact of the sampling rate on the anomaly detection performance according to the sampling algorithm, for the first scenario.

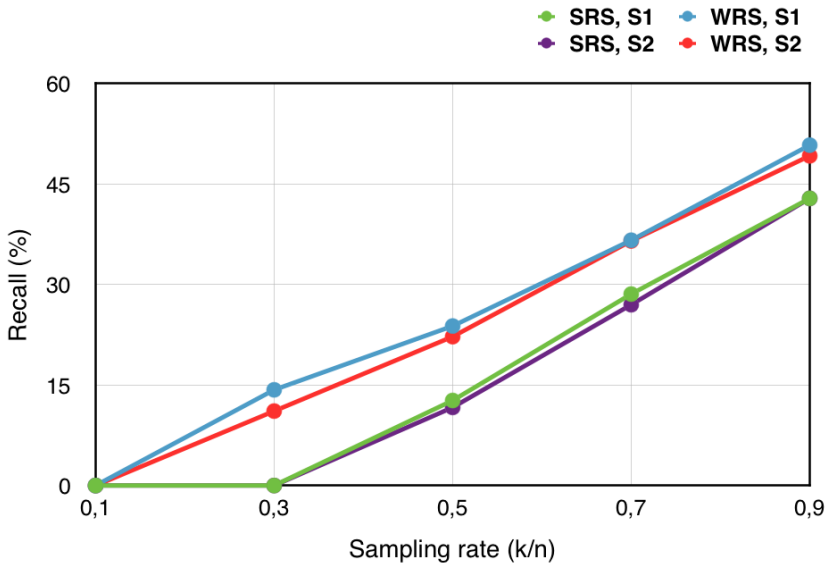


FIGURE 3.10 – Comparison of the sampling impact on the anomaly detection performance when using the Simple Random and Weighted Random Sampling algorithms for both scenarii $S1$ and $S2$.

3.4 Conclusion

Data streams represent a challenge to the data processing operations such as query execution and information retrieval. They pose many constraints in terms of memory space and execution time for the computation process. This is mainly due to the huge volume of the data and their high arrival rate. Generating approximate answers by using a small proportion of the data stream is acceptable for many applications.

In the first part of this chapter, we discussed three data streams sampling algorithms. Their goal is to maintain a representative, fixed-size sample of the most recent elements of the stream. At first, SRS and Deterministic sampling algorithms were adapted to the context of the sliding window. Secondly, the performance of these algorithms was compared to that of Chain-sample algorithm. The results of the experiments show that Chain-sample gives better results than SRS and Deterministic sampling in terms of execution time and sampling accuracy respectively.

The second part of this chapter was dedicated to studying the impact of data sampling on the anomalies detection results using EWMA algorithm. First, we explained the WRS algorithm that samples the data according to their values with respect to the values of their neighbors in the current window. Then, we sample the data using the three algorithms: Deterministic sampling, SRS and WRS. The obtained results showed that the WRS algorithm outperforms both Deterministic sampling and SRS algorithms in terms of the true positives rate.

Part II

Managing data quality in streaming sensors networks

Chapter

4 Modeling data quality in sensors networks

Contents

4.1 Introduction **69**

4.2 Data quality basic concepts **70**

4.3 Data quality management in sensor networks **74**

4.3.1 Data quality dimensions in sensor networks 74

4.3.2 Sensor data quality management, a new approach 79

4.4 Data quality in streaming sensor networks, related works . **80**

4.5 Conclusion **84**

The scientific contribution presented in this chapter has been published in our paper: **Rayane El Sibai**, Yousra Chabchoub, Raja Chiky, Jacques Demerjian and Kablan Barbar. *"Assessing and Improving Sensors Data Quality in Streaming Context"*. In the 9^{th} International Conference on Computational Collective Intelligence Technologies and Applications (ICCCI), pages 590-599, Springer, September 2017.

4.1 Introduction

Data quality plays an important role in the analysis of environmental data. An environmental monitoring process consists of regularly collecting and analyzing the data streams coming from sensors. It aims to infer new knowledge about the environment, allowing the network explorer to supervise the network and make the right decisions. Different data mining techniques are then applied to the collected data in order to derive useful statistics for the detection and prediction of the anomalies.

The data analysis results are closely dependent on the quality of the collected data. In the real world, the data are often dirty, they are noisy, erroneous, and contain duplicate and missing values. As in the data analysis process, the conclusions and decisions are based on the data, this leads to faulty and defective results if the data have poor quality.

This chapter is organized as follows. We present in Section 4.2 the basic concepts of data quality. We discuss in Section 4.3 several quality dimensions for sensors data, and we provide our definitions for the accuracy and confidence dimensions. We also propose a new model for data quality management in sensor networks. Compared to existing approaches, our model takes into account the errors caused by sensor defects in the evaluation and improvement processes of data quality. We debate in Section 4.4 the existing research studies related to data quality in streaming sensor networks. We end the chapter with a conclusion.

4.2 Data quality basic concepts

The amount of data we produce and consume every day is growing exponentially in the modern world. For instance, about $12TB$ of tweets on Twitter and $25TB$ of log data on Facebook are generated every day. Further, 30 billion RFID tags and 4.6 billion camera phones are used all over the world today. In addition, in 2011, there were 2 billion people connected to the web [Zaslavsky et al., 2013]. The data can be used and analyzed to improve the performance of systems and decision support applications as well as for risk assessment.

Karr *et al.* [Karr et al., 2006] define the data quality as follows:

"*Data quality is the capability of data to be used effectively, economically and rapidly to inform and evaluate decisions.*"

The interest in data quality comes from the fact that the decisions are based on the data. The decisions can have very serious consequences when the data are of poor quality. Improving the data quality is about improving the quality of decisions.

For instance, in the medical sector, some drugs can be taken off the market due to faulty side effects reported by the clients. A decimal point misplaced in the prescription of medicine was responsible for the death of a pediatric child [Belkin, 1997]. The healthcare organization states it pays about 4 million dollars per year following the claims of patients who became ineligible due to medical malpractice often due to inaccurate patient data [Madnick et al., 2005]. In the commercial sector, poor data quality can have quite negative economic impacts on both organizations and customers. Dirty data are reported to cost US industry billions of dollars each year [Fan, 2015]. For business companies, poor data quality impacts customers and employees satisfaction. Also, poor data quality requires more time and resources to be processed and cleaned, which increases their operating costs.

In the recent years, data quality management has attracted the attention of businesses and academic communities. This is manifested by the growing number of publications related to data quality during the last twenty years, as shown in Figure 4.1 [Moges, 2014].

The quality of the data does not only concern their accuracy but extends to many other dimensions such as completeness, objectivity, timeliness, representation, security, etc [Wang, 1998]. Nowadays, there are no standard methods for assessing data quality. This latter is defined as *"fitness for use"* [Neely, 2005]. Data quality dimensions and their evaluation, as well as their improvement methods, depend on the requirements and the needs of the users and applications. The data that may be considered good in one case may not be in another case. Therefore, the quality of the data depends on the context relating the use of the data rather than the data themselves.

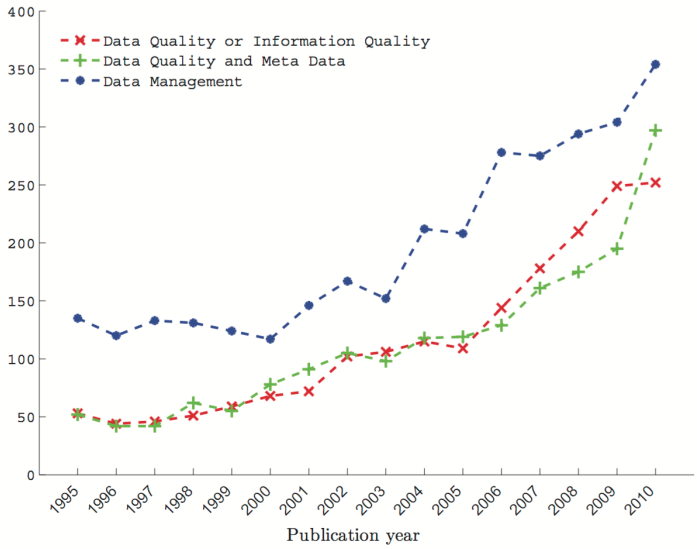


FIGURE 4.1 – *"Journal and conference proceedings from ISI Web of Knowledge searched by a query title and business economics domain using the key words information quality or data quality, data quality and metadata, and data management"* (From [Moges, 2014]).

Nowadays, the field of data quality management is a part of several application domains and different research topics, such as the e-government, healthcare, life sciences and Web data [Batini and Scannapieco, 2010]. The need for data quality management has led to the definition of different methodologies for data quality management. These methodologies concern quality dimensions, models, techniques, tools, and methodologies adapted to new types of data and information systems.

The dimensions are quantitative indicators defined and calculated based on the characteristics of the data, and at the base of which, the data quality is evaluated. The improvement techniques are a set of activities based on algorithms and procedures used to improve the quality of the data. In database management systems, the data is modeled and manipulated using a query language specific to the information system being used. These systems need to be expanded to represent and manage data quality issues, including dimensions [Klein et al., 2007]. Methodologies are a set of guidelines and techniques that define the procedure for managing data quality. Total Data Quality Management (TDQM), Total Information Quality Management (TIQM), Data Quality Assessment (DQA) and Information Quality Measurement (IQM) are examples of methodologies [Batini and Scannapieco, 2010]. The tools and frameworks are a set of tools and graphical interfaces made available to the user to allow him to manage the quality of the data.

Total Data Quality Management Program

The need of companies for a complete data quality management application has led to the definition of different methodologies for data quality management, among them the TDQM Program methodology [Wang, 1998]. TDQM consists of four phases, as shown in Figure 4.2. The first step is the definition of the data quality dimensions to be evaluated, analyzed and improved. The second step is the measurement of these dimensions following which a set of metrics (quantitative values) are produced. The analysis phase makes it possible to identify the sources and causes of the detected quality problems. Finally, several actions are taken during the improvement phase in order to enhance the data quality.

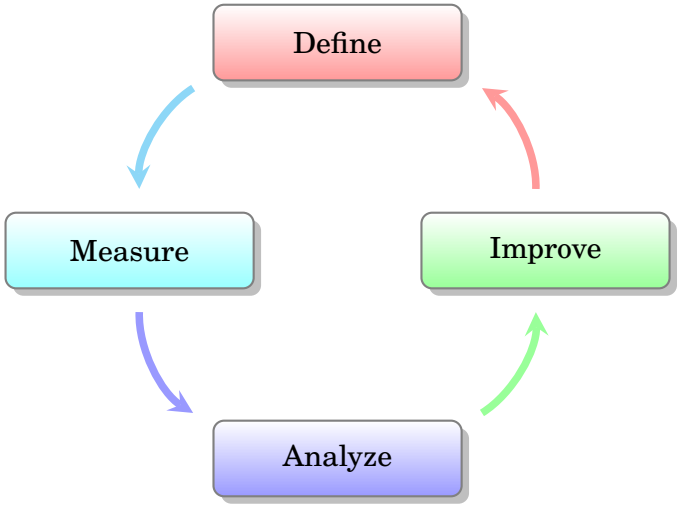


FIGURE 4.2 – TDQM methodology.

Data quality dimensions, empirical approach

The first step in managing data quality is the identification and definition of the dimensions on which the quality of the data will be evaluated. In fact, although the quality of the data is defined according to the context of its use, it is very difficult to measure the data quality without referring to a specific set of attributes or dimensions. The identification of the quality dimensions to be assessed and improved depends on the needs of the customers [Wang and Strong, 1996], while the assessment and improvement methods depend on the nature and characteristics of the data.

In the literature, the data quality dimensions can be defined according to three main approaches: theoretical, empirical and intuitive [Batini and Scannapieco, 2010]. We detail in the following the empirical approach.

In [Strong et al., 1997] and [Wang and Strong, 1996], a set of data quality dimensions in static environments was defined by interviewing data consumers. Four data quality categories have been proposed: intrinsic dimensions, contextual dimensions, representational dimensions and accessibility dimensions. Each one of these categories is divided into several dimensions as shown in Figure 4.3.

Intrinsic data quality dimensions measure the quality of the data value itself. Believability and Reputation dimensions represent the degree to which the user considers the data as correct and trustworthy. The estimation of the trust degree

requires verifying the provenance of the data (from which source it was generated) and the changes that it undergoes. For instance, Wikipedia's information has a poor reputation compared to those in the IEEE database. The third data dimension in the intrinsic category is accuracy. Accuracy qualifies the difference between the value stored in the database and the real value that the data aims to represent. The last dimension is data objectivity. It represents the degree to which the data are equitable and unbiased.

Contextual data quality category considers the context in which the data will be used. Relevancy dimension also known as helpfulness and domain decision describes the satisfaction degree of the user's needs and tasks. For instance, for the search engines, the pages returned following a user request are relevant if they contain the answer. Timeliness dimension also known as freshness represents the age of the data and it can be exploited in several ways.

[Wand and Wang, 1996, Redman, 1997] define the timeliness as the rapidity with which the data are updated in the relational table. [Wand and Wang, 1996, Liu and Chi, 2002] consider the timeliness as the ability of the data age to meet the application's needs. According to [Naumann, 2002], the timeliness represents the average of the data age in the warehouse. In this case, the age of the data does not mean the data antiquity regarding the current time, but rather the age of the last update. [Jarke et al., 1999] defines the timeliness dimension as the data volatility. It is the frequency with which the data values vary over time. For instance, the weather conditions have a high volatility since their values change frequently. Completeness dimension measures the size of the data, it is given by the ratio of the number of real-world data and the size of the tuples registered in the database.

Representational data quality category is used to capture the quality of the data representation. The first dimension is Interpretability. It represents the degree to which the data is clear, simple and appropriate for the user. It concerns the availability of the documentation required for an interpretation of the data. The second dimension is Ease of understanding. It depicts the degree to which the semantic relation between the different information is understandable by the user. The third dimension is Representational consistency, also known as homogeneity and value consistency. It represents the degree to which the data are compatible with the previous data. The last dimension is the Concise representation, known as structural consistency and format precision. It is the degree to which the data structure is suitable to the data itself.

The fourth and last data quality category is accessibility. It is related to the accessibility of the data and their security level. The first dimension is Accessibility, also known as availability. It is a technical criterion related to the connection between the user and the source of the data, it measures the probability that a user query is answered within a specific time range. The second dimension is the Access security dimension, known as privacy. It incorporates technical security aspects such as data encryption/decryption, user login, anonymization of the user and authentication of the data source.

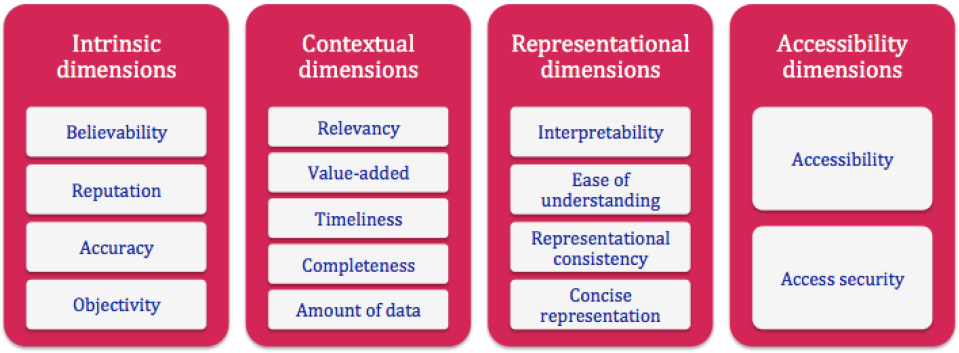


FIGURE 4.3 – Data quality dimensions, empirical approach [Strong et al., 1997].

4.3 Data quality management in sensor networks

In this section, we extend the data quality dimensions for sensors data. We choose the most important quality dimensions in view of their impact on the data analysis results: precision, accuracy and completeness. These dimensions will be discussed in the following. Notice that the fact that data are generated in the form of streams does not change the definition of these dimensions, but adds several complexity degrees to the data quality evaluation and improvement algorithms. These complexities are mainly caused by the infinite data stream size and its variable rate.

4.3.1 Data quality dimensions in sensor networks

In real-world such as sensors environments, data are often dirty, they contain noisy, erroneous, duplicated and missing values. This is due to many factors: local interferences, malicious nodes, network congestion, limited sensor precision, harsh environment, sensor breakdown, sensor malfunction, miscalibration and insufficient battery power of the sensor. As in the data analysis process, the conclusions and decisions are based on the data, this leads to defective and faulty results.

One solution to overcome this problem is to use sensors with high precision to could assume that the arising errors are small, and to deploy redundant sensors to cover the breakdown of a given sensor. Nevertheless, this approach is very expensive as it requires very high costs for the sensors. That is why we opted for a software-based solution where we evaluate and improve data quality using several complementary methods.

4.3.1.1 Precision dimension

Data precision is a measure of random noise [Smith, 2013]. It depicts how close are the data values to each other. The potential causes of noisy data are the fluctuations and interferences in the environment, the low battery of the sensor, the hardware failure and the poor calibration of the sensor [Ni et al., 2009]. As a result, the data can be influenced by *random errors* also called *noise*, which makes them slightly deviate from the true values. Random errors are always present in the data and cannot be controlled. They impact the variability of the data around the average

without affecting it. Thus, the data values will be scattered and dispersed around the true values. Keeping the noise in the data can have a very serious impact on the data analysis and queries results. Therefore, it is necessary to detect and to remove these errors [Elnahrawy and Nath, 2003, Tan et al., 2005] in order to extract the relevant information from the data.

The commonly statistical measure used to detect the noise is the variance [Ni et al., 2009, Sharma et al., 2010, Abuaitah and Wang, 2015]. Actually, a high variance is a sign of noisy data. However, evaluating the variance of the data without relating it to the mean is useless. Indeed, a set of data having a standard deviation $\sigma = 5$ and a mean $\mu = 6$ does not have the same interpretation as a set of data having $\sigma = 5$ and $\mu = 100$. That is why the Coefficient of Variation (CV) [Smith, 2013] is more efficient than the standard deviation to detect the noise. CV is defined as the ratio of the standard deviation σ to the mean μ [Lohninger, 2010]. A lower value of CV implies a good data precision [Smith, 2013]. According to the value of CV, data precision can be considered as:

- Good: In this case, the data is kept and no improvement will be applied.
- Medium: In this case, the data have to be denoised.
- Bad: In this case, the data have to be deleted as it contains much more noise than useful information.

When the data is to be denoised, smoothing algorithms can be applied. In fact, the smoothing process reduces the data variance, and therefore, attenuates the contribution of the noise in the data.

4.3.1.2 Accuracy dimension

The accuracy dimension describes how close are the readings to the real observations. It represents the difference between the observation's value and the true value which the sensor reading aims to represent. Due to instrumental, physical and human limitations, malfunction and miscalibrating of the sensor, the observations values can deviate significantly from the true values. These deviated values called *errors* are abnormal compared to other data and they affect the average of the data. Actually, a deviant value compared to other data or to the expected value is considered as *abnormal*.

As shown in Figure 4.4, abnormal data may represent an anomaly in the sensor or an anomaly in the environment. In the first case, it is a false measure called *error* or *fault* and must be removed and replaced to avoid its influence on the data analysis process. In the second case, it is a real measure called *event* that describes a real-world phenomenon which must be exploited. The detection and the removal of erroneous data help the network explorer to improve the data accuracy. Therefore, in order to ensure a good data accuracy, we propose to separate the true data from the erroneous ones caused by sensors faults, so that only the true data will be kept.

One way to separate errors and events is to model the events that may occur in the environment by defining their characteristics. Thus, the features of the abnormal data will be compared to those of the predefined events in order to classify the abnormal data into errors and events. However, in a variable dynamic environment such as sensor networks, it is not always possible to predict the events that can occur.

An alternative solution is to define the errors, also called sensors faults, and classify the data based on the characteristics of these errors. We adopt this solution in the following, and we study on the basis of a literature review the various types of errors that can occur in sensor networks.

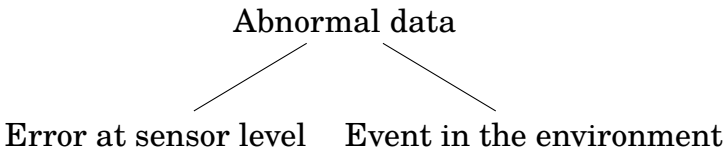


FIGURE 4.4 – Types of abnormal data.

Taxonomy of data faults

In the following, we discuss the faults encountered in sensor networks. The faults can be examined from two points of views: data-centric view and system-centric view [Ni et al., 2009]. The data-centric view uses the features of the data (mean, variance, etc) to describe the faults, while the system-centric view consists on monitoring the hardware aspects of the sensors and how they affect the data features [Ni et al., 2009].

Since some faults, such as the outliers, have unknown causes, it will be easier to model the fault based on the data features rather than the hardware aspect [Ni et al., 2009]. Table 4.1 describes the faults that can occur in sensors environments from a data-centric view.

TABLE 4.1 – Taxonomy of sensor data faults from a data-centric view [Ni et al., 2009].

Fault	Form of occurrence	Potential cause(s)	Impact on
Outlier	Single point	Unknown	Gradient
Spikes	Successive points	Low battery Battery failure Sensor failure Connection failure	Gradient
Stuck-at x	Successive points	Low battery Dead sensor Sensor ADC malfunction	Variance

- Outliers: A sudden and temporary increase or decrease in the sensor values is manifested by the appearance of outliers [Ni et al., 2009, Abuaitah and Wang, 2015], also called Short faults in [Ramanathan et al., 2006, Sharma et al., 2010]. An outlier is an isolated observation that deviates significantly from the other observations.

Several real-world deployments of sensor networks reported the presence of such an error in the data. One can cite [Szewczyk et al., 2004, Ramanathan et al., 2006, Ingelrest et al., 2010]. An example of such error is given by Figure 4.5. The detection of outliers can be based on a set of rules that need to be satisfied by each data instance. These rules are defined according to the application domain.

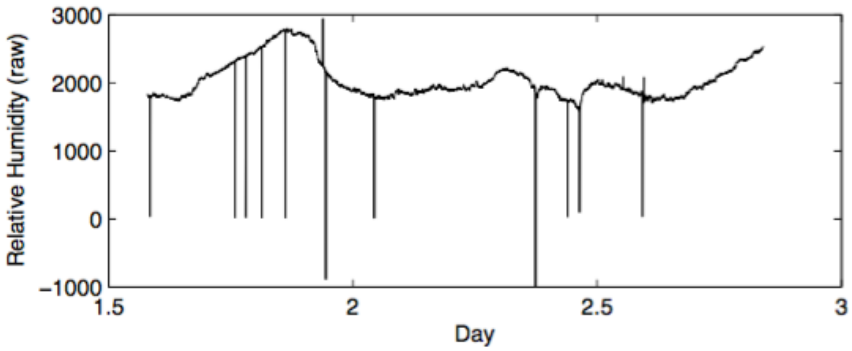


FIGURE 4.5 – Examples of outliers faults in the raw humidity readings in the NIMS deployment [Kaiser et al., 2005].

- Spikes: Spikes errors are popular in sensor networks. A spike error consists of a set of points that deviate significantly from the expected ones. Notice that an outlier can be considered as a single-sample spike [Sharma et al., 2010]. This type of fault is characterized by a sudden and large increase of the gradient [Ni et al., 2009, Abuaitah and Wang, 2015].

- Stuck-at: A stuck-at x error occurs when the sensor is stuck on an incorrect value x. During such a situation, a set of successive values will have the same value $x \pm \epsilon$. The error may last for a long time, and the sensor may or not return to its normal behavior [Ni et al., 2009]. This type of error is also called "CONSTANT" in [Sharma et al., 2010, Ramanathan et al., 2006]. According to [Sharma et al., 2010], this type of error affects about $20\% - 25\%$ of the data of the INTEL Lab and $15\% - 35\%$ of the data in NAMOS data set. An example of such error is given by Figure 4.6.

  Stuck-at data are not always errors. This is the case of Clipping situation [Ni et al., 2009]. The data that exhibit a Clipping fault still hold some real and important information, and should not be discarded. Clipping errors occur when the environment data are outside the maximum sensitivity range of the sensor. In this case, we observe consecutive data having the maximum value of the sensor sensitivity range. The sensor is thereby saturated, and the data exhibiting this fault have a very small variance [Ni et al., 2009].

  The detection of stuck-at errors will be discussed in Chapter 5.

The accuracy dimension for each sensor reading takes the following values:

$$Accuracy = \begin{cases} 0, & if\ the\ value\ is\ erroneous. \\ 1, & if\ the\ value\ is\ not\ erroneous. \end{cases}$$

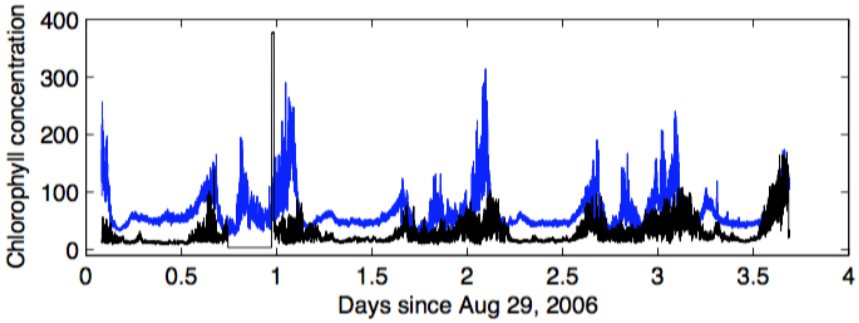


FIGURE 4.6 – Stuck-at faults in the chlorophyll concentrations from two buoys in the NAMOS deployment at Lake Fulmor monitoring the marine environment [Dhariwal et al., 2006].

Sensor reliability dimension: Based on the accuracy dimension, we define the sensor reliability dimension. The sensor reliability degree, denoted by $cumulativeError$, depicts the percentage of the cumulated erroneous values registered by the sensor. At each instant t (in $hour$), the $cumulativeError$ degree is calculated as follows:

$$cumulativeError_t = [\frac{Number\ of\ erroneous\ data_t}{Number\ of\ received\ data_t}] \times 100\%$$

4.3.1.3 Completeness dimension

The loss of data has serious consequences for the environmental monitoring system and leads to reduced information, and so, to erroneous and distorted results. One solution to deal with this problem is to use a reliable transmission protocol [Pang et al., 2008], which requires data retransmission in case of failure. However, this will be costly in terms of battery consumption. Another way to avoid data loss is to deploy multiple sensors in the same region to be monitored. This approach is not only expensive in terms of hardware, but it will also create a problem of data redundancy. In fact, it requires an effective policy to merge the data and will increase the data preprocessing duration before their analysis. We can thus conclude that the most efficient way to deal with the problem of missing data is to estimate and regenerate them using statistical methods.

The completeness dimension at each instant t (in $hour$) is measured in terms of the $cumulativeMissing$ degree calculated as follows:

$$cumulativeMissing_t = [\frac{Number\ of\ missing\ data_t}{t \times streamRate}] \times 100\%$$

where $streamRate$ is related to the data frequency of the considered stream.

Several approaches have been proposed in the literature to address the problem of missing data. One can cite the imputation by regression [Cool, 2000], hot-deck imputation [Iannacchione, 1982], Expectation Maximization (EM) [McLachlan and Krishnan, 2007], maximum likelihood [Little and Rubin, 2014] and multiple imputations [Rubin, 2004]. However, due to the constraints related to data streams environments as well as the temporal and spatial characteristics of sensors data, the use of these methods becomes unsuitable. Thus, new methods have been developed.

[Gruenwald et al., 2007] proposed to use the spatial correlation between sensors to impute missing data for a specific sensor. The proposed solution, so-called Freshness Association Rule Mining (FARM), uses the association rule mining to find the relationships between the sensors while taking into consideration the freshness of the data. The more recent the data, the higher is its weight during the estimation. For a given sensor having missing values, the imputation of missing data is done by weighting the average of the values of its related sensors. Pan *et al.* [Pan et al., 2010] designed a K-nearest neighbor algorithm to estimate missing data in sensor networks. The algorithm uses a linear regression model based on the temporal and spatial correlations of the sensors to impute missing data. [Li and Parker, 2008] developed a fuzzy adaptive neural network that uses both temporal and spatial correlation of the data to estimate the missing values. For a given sensor having missing data, Window Association Rule Mining (WARM) [Le Gruenwald, 2005] uses the association rule mining to find its related sensors, and the values reported by the related sensors in the last sliding window are used to estimate the missing values.

4.3.1.4 Confidence dimension

We define the $confidence$ dimension to be the degree of trustiness that we give to a particular sensor reading after evaluating and improving (if necessary) its quality. The $confidence$ degree depends on the originality of the data. Actually, when a value is missing or erroneous, it has to be regenerated in order to enhance its quality. In this case, its $confidence$ degree is equal to $x\%$ which represents the proportion of consecutive missing or erroneous values around it. x is calculated relatively to given time period (24 hours as an example). Therefore, the $confidence$ degree of each sensor reading takes the following values:

$$Confidence = \begin{cases} x\%, & if\ the\ value\ is\ regenerated. \\ 100\%, & if\ the\ value\ is\ original. \end{cases}$$

4.3.2 Sensor data quality management, a new approach

Based on the previous study, we propose the following new strategy to manage sensors data quality. Our solution is shown by Figure 4.7, it is based on the TDQM methodology. Recall that this methodology consists of a series of iterations to manage the data quality. These iterations are the definition of data quality dimensions, the measure of these dimensions, the analysis and the identification of the causes of poor data quality, and finally, the improvement of the data quality.

After the data acquisition from each sensor, the system assesses the precision, accuracy, sensor reliability and completeness dimensions. Based on the obtained measurement, it detects noisy data, detects erroneous data, and finally, detects missing and duplicated data. Thereafter, it proceeds to smooth noisy data using smoothing algorithms, to remove duplicated data, and finally, to remove and to regenerate erroneous data and missing data. Afterward, for each data, the confidence dimension is evaluated. Finally, the information related to the data quality dimensions will be attached to the data and propagated to the user.

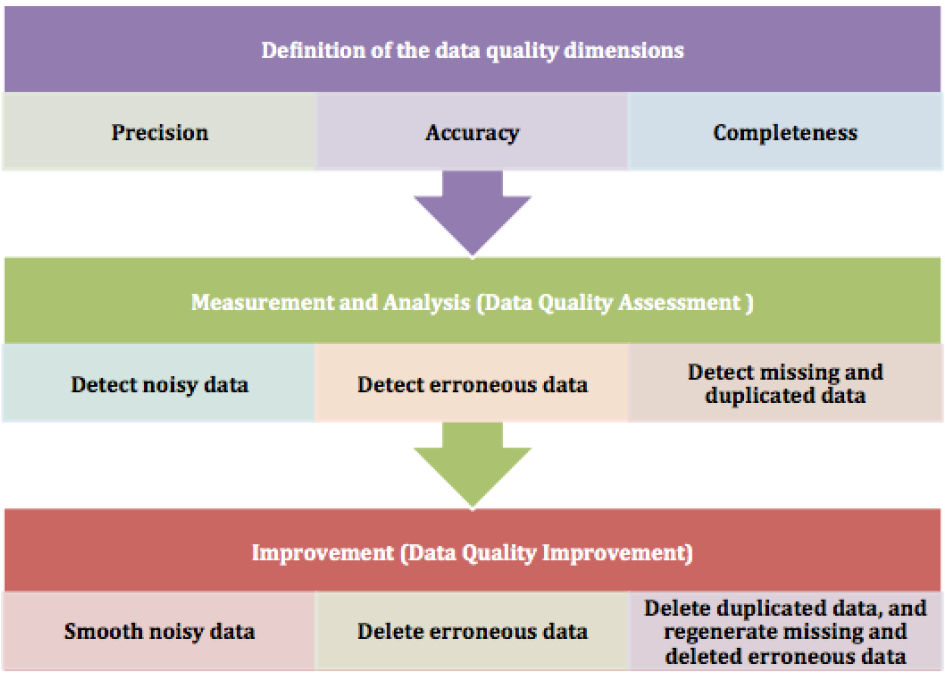


FIGURE 4.7 – Sensors data quality management approach based on TDQM methodology.

4.4 Data quality in streaming sensor networks, related works

In this section, we discuss the existing approaches to manage data quality in streaming sensor networks.

A sensor data cleaning framework called Extensible Sensor stream Processing (ESP) has been proposed in [Jeffery et al., 2006] to clean sensor data. The framework uses continuous query language (CQL) as a declarative language to clean the data based on the user's rules. CQL is a cleaning process that aims to detect outliers data, replace missing data and remove duplicate data. The detection of erroneous data is achieved through two steps. First, the framework ESP considers that the data is erroneous if it is largely higher than the expected value defined by the user. This expected value depends, of course, of the application domain. Second, ESP uses the spatial correlation between the data. It compares each sensor value with the ones recorded by its neighboring sensors. If the difference between the value and the mean is two times higher than the standard deviation, the data is considered as erroneous. For the missing data problem, ESP replaces each missing value by interpolation. Although this work is interesting, it remains incomplete. In fact, a deviated data can be an erroneous data or an event. The outlier detection process of ESP does not make this differentiation. Also, no detection and removal of noisy data were performed.

Two sensor data cleaning approaches were proposed in [Zhuang and Chen, 2006]. These approaches only consider the accuracy quality dimension and are intended to detect, delete and/or correct the erroneous data. These errors can be of two types: simple short outlier and simple long outlier. The first approach uses wavelets to

correct erroneous data, while the second approach is based on similarity comparison based on the neighboring dynamic time warping (DTW) distance to detect and to delete erroneous data.

Bettencourt *et al.* [Bettencourt et al., 2007] proposed a statistical method for detecting errors and events and for replacing missing data in a wireless sensor network deployed at Sevilleta National Wildlife Refuge. Under the assumption that the spatio-temporal correlation degree between sensors is high, the method requires learning statistical distributions of differences between the readings of a given sensor over time, and between the readings of a given sensor and the readings of its neighbors. On the basis of the learned statistics, the likelihood probability for each new sensor reading is calculated and the classification of the observation as error or event is made. The main concern with this approach is that it does not take into account the erroneous data caused by sensor faults.

The origin of the data was used to assess the accuracy dimension of both sensors and data in [Lim et al., 2009]. The proposed framework computes the sensor and data confidence degrees and updates them over time. A filtering process is performed based on a confidence interval defined by the user. The objective is to select only the data having a confidence degree belonging to the confidence interval. For the computation of data item confidence degree, two similarity measures are used: the value similarity and the provenance similarity. To evaluate the similarity, the authors assume that the theoretical data distribution follows the normal distribution. The confidence degree is then calculated using the difference between the experimental and the theoretical distribution. The provenance similarity is based on the fact that if several sensors record the same value, this increases the confidence degree of this value. However, in many real use cases, no a priori knowledge about the data distribution can be defined. Moreover, the framework does not present any way to improve the data accuracy.

A data fusion system based on data quality in a distributed context was proposed by [Hermans et al., 2009, Hermans, 2009]. The system uses a set of heuristics to evaluate the data quality dimensions, namely, accuracy, precision, completeness and timeliness dimensions. The network is organized in clusters, and in each cluster, a sensor node is chosen as a cluster head. Each sensor node evaluates its data quality and sends the results to its cluster head which will merge the received data while taking into account their qualities.

A data cleaning system based on machine learning algorithms was proposed in [Ramirez et al., 2011]. The system aims to evaluate the accuracy dimension of the data generated from the environmental sensor network Jornada Experimental Range (JER). For every sensor, each value is predicted using three machine learning algorithms. If the recorded value is far from that predicted, the value is declared as an error and will be replaced by the expected value or by the average of the erroneous value and the expected value or by interpolation. According to the author, this approach is very expensive and so it will be implemented as a post-processing phase. This needs to save all the recorded values, which is not suitable for the streaming environment constraints. This work presents several weaknesses. Firstly, as with previous works, no distinction between the different types of outliers has been taken into account, a value very different from that predicted may simply represent a real event and so should not be replaced. Also, the completeness dimension was not

addressed, while it is quite important in sensors environments as the transmission conditions are very variable and the data are very likely to be missed.

According to [Gutiérrez Rodriguez, 2010, Rodriguez and Servigne, 2012], sensors data quality depends on the quality of the data sources and the quality of the treatment that the data have undergone. [Gutiérrez Rodriguez, 2010, Rodriguez and Servigne, 2012] developed MoSDaQ prototype: an environmental phenomena monitoring system for volcanic data analysis. The system includes three data management layers, in the acquisition layer, the sensors' observations are recorded. In the processing layer, the data are analyzed, they are filtered, reduced or aggregated. Finally, the discovery layer allows the user to exploit the data in real-time. The data quality characteristics will be provided to the user via a graphical user interface. The evaluation criteria for the data quality are the accuracy, the completeness, and the time-related aspect. To evaluate the accuracy dimension, no differentiation between errors and real data has been taken into account.

[Li et al., 2012] defined three quality dimensions for real-world data. These dimensions are the currency, availability, and validity. The currency dimension represents the utility of the data in relation to its time. The more recent the data, the more is considered reliable and representative of the real world. The data is considered available to users as long as it is not expired and can respond to user requests. The percentage of time that data is reliable and not expired is represented by the availability dimension. Finally, the validity dimension evaluates the accuracy of the data and is evaluated by a set of static and dynamic rules. This metric depends largely on the field of application and the considered scenario. For instance, "The temperature in Vienna should be between 0 and 35 degrees in May" is a static rule. However, the violation of these rules can be due either to real data representing a particular phenomenon in the environment which needs to be exploited, or an erroneous data caused by a sensor defect. This differentiation is not taken into account in the definition of the validity dimension proposed in this work.

In [Islam et al., 2014], the authors studied the impact of missing and incorrect data on the classification of data issued from sensor networks. They proposed a strategy to improve the data quality based on two dimensions: completeness and accuracy. The strategy consists of identifying and removing erroneous data and then imputing the deleted erroneous data and missing data. The adopted method for the identification of incorrect data is based on the so-called Co-appearance based Analysis for Incorrect Records and Attribute-values Detection (CAIRAD) technique (see [Rahman et al., 2012] for more details) and does not consider the impact of sensor hardware faults on the data correctness.

In order to improve the data reliability and to reduce the energy consumption of sensors, Lei *et al.* [Lei et al., 2016] proposed to clean sensors data. The key point of the proposed strategy is the outliers detection and their classification into errors and events. At first, the observations of a given sensor are predicted using a linear regression model. Secondly, each observation is considered abnormal if the difference between its value and the predicted one exceeds a certain predefined threshold. Finally, the classification of each detected abnormal observation into error and event is based on the Euclidean distance between the observation value and that of the nearest neighbor of the sensor. The abnormal observation is considered as erroneous if the Euclidean distance exceeds a certain predefined threshold. [Tasnim

et al., 2017] proposed a framework for cleaning sensor data. It aims to evaluate the credibility dimension of the data. The credibility of the data measures the number of times a sensor has recorded the value correctly in relation to the number of observations recorded by the sensor during a certain period of time. For a given sensor, the difference between each recorded value and the predicted one is calculated. The predicted value is the average value of the readings of the spatial neighbors of the sensor. If the difference exceeds a certain threshold, the recorded value will be considered as erroneous. However, the deviation of a sensor reading from the spatial neighbors' values may represent a real phenomenon occurring in the environment and not an error. This differentiation was not taken into account by these two works.

[Cheng et al., 2018] presented a new approach for cleaning sensor data. They proposed to evaluate the data quality based on four dimensions: data volume, accuracy, completeness, and timeliness. Based on correlation level between these dimensions, several strategies for data cleaning are executed. The proposed approach does not consider the erroneous data caused by sensor faults in the evaluation of the accuracy dimension. A sensor reading is considered erroneous only if the difference between its value and the real value exceeds a certain threshold.

Few works have elaborated the quality of the sensors data in streaming context. In [Klein et al., 2007], the authors proposed a model to evaluate and store information about data accuracy and completeness dimensions. These information are recorded over jumping windows instead of each item in order to reduce the storage size. According to the authors, a data is considered as inaccurate only when it has a value exceeding the highest sensor range. Data accuracy is evaluated according to the precision of the sensor and does not consider the errors caused by sensors defects. Moreover, no improvement of data quality was proposed. The previous work [Klein et al., 2007] was extended by [Klein and Lehner, 2009, Olbrich, 2010]. [Klein and Lehner, 2009] studied the impact of several data stream operators on the quality of the data. Data stream operators are part of data manipulation and aim to manage the data by applying modifying, generating, reducing and merging actions. [Olbrich, 2010] provided the information collected about the quality of the data to the user. SQL functions are implemented in order to control the quality of the data on each window and to return to the user only the data having a quality degree that meets his requirements. However, just like the initial work, no data quality improvement was proposed. The rules used to evaluate the accuracy of the data do not differentiate between errors and real readings.

Errors identification in the raw data is a very important issue, it separates the erroneous data and the true data, and thus, ensures a good data accuracy. In the most of the studies presented above, the identification of erroneous data was always linked to the precision of the sensor (do not confuse with the precision dimension related to noise) and without making any difference between the real data and the erroneous ones caused by sensor faults. Also, few approaches addressed at the same time the three dimensions: precision, accuracy and completeness, and few studies plan to enhance the quality of sensors data.

4.5 Conclusion

In this chapter, we described the basic concepts of data quality management and we focus on the data quality dimensions in sensor networks. In order to avoid faulty decision due to dirty data, data quality must be guaranteed. There are two approaches to handle and to improve data quality deficiencies in sensors environments:

- Hardware-based approach: It consists of the use of sensors with high precision and assuming that the arising errors are small, and the use of redundant sensors to cover the breakdown of a sensor. This approach is very expensive as it requires very high costs for sensors.
- Software-based approach: To prevent high costs while guaranteeing a good data quality, data quality information have to be recorded and processed and the data quality has to be improved. Therefore, it is necessary to evaluate and to improve the quality of the data before exploring them. This is the key function of a sensor data quality management system.

In the literature, there are several software-based approaches for data quality management in sensor networks. However, in the most of them, errors identification in the data was not taken into consideration. Errors identification is a very important step since it separates erroneous data and real data, and thus, guarantees a good data accuracy. A part of the proposed system in Section 4.3.2 is implemented and will be presented in Chapter 7. We discuss in the next chapter the detection of erroneous data of type stuck-at.

Part III

Data streams anomalies detection

Chapter

5 An in-depth analysis of CUSUM algorithm for change detection in time series

Contents

5.1 Introduction **88**

5.2 Anomalies detection algorithms **88**

5.3 An analysis of CUSUM algorithm **91**

5.3.1 Algorithm description 91

5.3.2 Choice of the parameters 92

5.3.3 Variability of the Run Length (RL) 94

5.4 Enhancing the reactivity of CUSUM algorithm **95**

5.4.1 Detecting the anomaly start time 96

5.4.2 Detecting the anomaly end time 97

5.5 Detecting mean change **97**

5.5.1 Efficiency metrics 97

5.5.2 Experimentations 99

5.6 Application to stuck-at error: Detecting variation change . **101**

5.7 Conclusion **104**

The scientific contribution and the obtained results presented in this chapter have been published in our paper: **Rayane El Sibai**, Yousra Chabchoub, Raja Chiky, Jacques Demerjian and Kablan Barbar. *"An in-depth analysis of CUSUM algorithm for the detection of mean and variability deviation in time series"*. In Proceedings of the 16^{th} International Symposium on Web and Wireless Geographical Information Systems (W2GIS), Springer, 2018.

5.1 Introduction

Assessing and improving the quality of the data in sensor networks poses significant challenges. Indeed, the data recorded and sent by the sensors are often dirty, they contain noisy, erroneous, duplicate and missing values. This may be due to many reasons, such as a sensor malfunction, an uncalibrated sensor, a low sensor battery, or caused by external factors such as the weather conditions, interference, etc.

In this chapter, we study the anomalies detection in the time series emitted by sensors. In particular, we discuss the slow and gradual changes in the process variability as they illustrate deviations in the calibration of the sensor, the so-called stuck-at error. For that, we present in this chapter an in-depth analysis and an improvement of the reactivity of CUmulative SUM (CUSUM) algorithm since it is well adapted to the detection of small deviations. Firstly, we discuss the choice of CUSUM parameters in order to optimize its results according to the compromise between the false positives and the Average Run Length (ARL) required by the algorithm to detect a deviation of the process mean. A study of the variability of the Run Length (RL) is provided by simulation. Secondly, we present an efficient method for estimating the start time and end time of the process mean deviation in order to improve the reactivity of CUSUM algorithm. Finally, we adapt CUSUM to detect a deviation of the process variability. All these improvements are validated by simulation and against real data streams.

This chapter is organized as follows: In Section 5.2, we discuss several anomalies detection methods. In Section 5.3, we study by simulation the choice of the parameters of CUSUM algorithm as well as the RL variability. In Section 5.4, we propose an efficient method for the estimation of the start and end times of the process mean deviation. The efficiency of the proposed approach is discussed in Section 5.5. The adaptation of CUSUM algorithm to detect the stuck-at errors is presented in Section 5.6. We end the chapter with a conclusion.

5.2 Anomalies detection algorithms

Data streams have significantly increased with the development of the Internet of Things (IoT) and the emergence of connected objects. Large and heterogeneous collections of data are continuously generated by these objects at a high rate. They are issued from the activity of different organizations belonging to various domains such as healthcare, financial services, social networks, logistics and transport, and public administration. Detecting online a change of the process parameters in data streams is an important issue as it can have several interpretations depending on the application domain. In [Chabchoub et al., 2014], the authors showed that an abrupt increase in the number of destination ports in IP traffic is an efficient criterion to detect port scan attacks. In [Manonmani and Suganya, 2010], an analysis of data streams issued from a multi-temporal satellite is provided. It aims to detect land use and land cover changes. Such changes can be explained by human activities, natural conditions and development activities.

Dunning *et al.* [Dunning and Friedman, 2014] define the anomalies detection as follows:

"*Anomaly detection is based on the fundamental concept of modeling what is normal in order to discover what is not.*"

Several anomalies detection methods have been designed by the statistics research community. According to [Basseville et al., 1993], these methods can be classified into two categories. First, we distinguish univariate change detection techniques. In this category, one can cite the control chart methods including Shewhart, Exponential Weighted Moving Average (EWMA) [Roberts, 1959], GEOMetric Moving Average (GEOMMA) [Roberts, 1959], CUSUM algorithm, and Bayes-type algorithms. The second category concerns univariate change detection algorithms. They are adapted to more complex changes such as non-additive changes in multidimensional signals. Among these methods, one can mention the time-series modeling and forecasting models such as Autoregressive Moving Average (ARMA) and Autoregressive Integrated Moving Average (ARIMA) models, and the likelihood ratio. Several criteria can be considered to compare all these methods. The comparison can be based on the tolerance to false positives (false alarms), the response time of the algorithm (Run Length to detect the change), or the adaptation to progressive (or small) changes. The choice of the suitable change detection method depends on the application field and the specificity of the targeted error or change.

Control charts are one of the main used tools used in Statistical Process Control (SPC). SPC is a set of techniques and tools for monitoring the quality of a process. The statistical control chart concept was first introduced by Walter A. Shewhart of the Bell Telephone Laboratories in 1924. Control chart algorithms are particularly used to monitor the process stability over time by detecting a change in its parameters. The type of the control chart depends on the number of process characteristics to be monitored. Basically, there are two types of control charts. The first one is called univariate control chart, it is a chart of one quality characteristic. The second type, called multivariate control chart, is a chart that represents more than one quality characteristic.

Control charts algorithms raise an alarm when the process presents a suspicious deviation from a standard behavior. This deviation is defined based on two given thresholds called control limits. In general, the chart contains three elements, the plotted data corresponding to the process itself, the control limits and a central line (process average). By comparing the plotted data to these control limits, we can deduce a decision about whether the process is stable (in control) or is unstable (out of control). A control chart consists of two phases. In phase I, called learning phase, the in-control process parameters are estimated and are used to define the control limits. In phase II, the control chart detects the changes in the process parameters. As long as the process remains in control, the data points fall within the control limits. If a data point falls outside the control limits, we consider that the process is out of control. An investigation is needed to find and eliminate the cause(s) of the occurred change. The main performance measure of a control chart is the ARL. RL is the number of observations required by the control chart algorithm to raise an alarm. When the process is in-control, the RL refers to false positives rate, and in the case of change, it qualifies the response time (reactivity) of the algorithm.

One of the most common control chart algorithms used to control the process

average is the Shewhart. Shewhart evaluates the state of the process based only on the information concerning the last observations of the process while ignoring the information provided by the past observations. It is efficient for the detection of large shifts, as it has a shorter response time than CUSUM in this case. However, Shewhart control chart is insensitive to small changes. CUSUM chart overcomes this problem by using the current observed data and the historical observed data. It cumulates the impact of small deviations over time which enables it to detect small shifts in the process mean. CUSUM algorithm was initially proposed by Page in 1954 [Page, 1954] and used to detect a shift in the process parameters. It aims to monitor the variation of the average of a process, and has the ability to detect small shifts (less than 1.5σ, where σ is the standard deviation) from the expected average (see [Montgomery, 2007] for more details). CUSUM control chart has been addressed in several research studies, such as [Ewan, 1963], [Bissell, 1969], [Goel and Wu, 1973], [Reynolds, 1975] and [Lucas and Crosier, 1982a]. In [Van Phuong et al., 2006], the authors propose to use CUSUM to detect attacks in sensor networks such as wormholes, sinkholes, hello flooding, and jamming. [Peng et al., 2007] use CUSUM algorithm to detect the denial-of-service attacks in the network. Reynolds *et al.* [Reynolds and Stoumbos, 2010] discuss the problem of CUSUM robustness to non-normality when monitoring the process mean and variance.

Another control chart designed to detect a change in the process mean is the EWMA introduced by [Roberts, 1959]. It is based on a weighted average that is updated for each received item of the stream. This moving average takes into account the current item and all the observed data. More importance (higher weight) is assigned to recent data. EWMA control chart has been addressed in many research studies, in particular, in the Statistical Process Control (SPC) domain. [Chabchoub et al., 2014] used EWMA to design an algorithm for on-line port scan attacks detection. They, first applied the sliding HyperLogLog algorithm to infer relevant statistics from the IP traffic data and then detected the malicious IP traffic data using EWMA control chart algorithm. Several research studies addressed the problem of process parameters estimation when phase I of the control chart contains data with anomalies. These data influence the in-control process estimated parameters, such as the mean and the standard deviation, and thus, makes the phase II less reliable to detect the occurred changes in the process parameters. [Zwetsloot et al., 2014] studied the effectiveness of several parameters estimation methods of the in-control process when phase I contains anomalies. [Zwetsloot et al., 2015] proposed to apply EWMA control chart in phase I in order to detect and to delete contaminated data so that the in-control process parameters estimation be robust. Saleh *et al.* [Saleh et al., 2015] studied the performance of EWMA control chart in terms of the mean and Standard Deviation of the ARL ($SDARL$).

Anomaly detection using the ARIMA model has been proposed by [Chen et al., 2005, Pena et al., 2013]. [Yu et al., 2016] introduced an improved version of ARIMA model for anomaly detection in wireless sensor networks. The complexity of detecting anomalies in multivariate time series has been addressed by [Tsay et al., 2000, Galeano et al., 2006]. Kalman filtering is another common method for anomalies detection in data streams. [Soule et al., 2005] proposed a method for the detection of traffic anomalies based on the Kalman Filter. At first, the matrix of traffic data is predicted. After that, the actual traffic matrix is estimated based on the recent traffic data. The difference between the actual traffic matrix and the predicted one

is computed and examined in order to detect the traffic anomalies. [Knorn and Leith, 2008] used a framework based on the Kalman Filter to monitor the software appliance. A mathematical model based on the Kalman Filter was proposed by [Manandhar et al., 2014] to detect attacks and faults on the smart-grid system. At first, the Kalman filter, combined with a mathematical model, estimates the state of the variables using the sensors readings. After that, the residual between the measured data and estimated ones is calculated, and the statistical test χ^2 is applied to check whether the residual corresponds to anomalies or not.

5.3 An analysis of CUSUM algorithm

5.3.1 Algorithm description

Applied on a data stream, CUSUM algorithm takes into account all the past values of the stream by calculating the cumulative sum of the deviations from the target value which is defined as the mean of the observables in the training window, μ_0. It is implicitly assumed that the observed process $(S_t)_{t\geq 0}$ is in control (has a standard behavior) during the training window. The cumulative sum control chart C_t, initially set to 0, ($C_0 = 0$), is calculated as follows:

$$C_t = \sum_{j=1}^{t}(s_j - \mu_0); t \geq 1$$

which is equivalent to:

$$\begin{aligned} C_t &= (s_t - \mu_0) + \sum_{j=1}^{t-1}(s_j - \mu_0) \\ &= (s_t - \mu_0) + C_{t-1}; t > 1 \end{aligned}$$

In order to quantify and detect small variations, CUSUM defines two statistics C_t^+ and C_t^-. C_t^+ accumulates for relatively high values of the observed process, the distance to $(\mu_0 + K)$; K being a given threshold that will be discussed later. For small values of the observed process, the cumulative distance to $(\mu_0 - K)$ is handled by C_t^-.

$$\begin{aligned} C_t^+ &= max[0, s_t - \mu_0 - K + C_{t-1}^+] \\ C_t^- &= max[0, \mu_0 - s_t - K + C_{t-1}^-] \end{aligned}$$

The threshold K is also called the allowance value. It depends on the mean shift that we want to detect. If either C_t^+ or C_t^- exceeds the decision threshold H, the process is considered as out-of-control and an alarm will be reported. The process is declared as in-control when the cumulative sum is again under the threshold H. K and H are often related to the standard deviation σ_0 calculated in the training window:

$$K = k\sigma_0;\ H = h\sigma_0$$

After the detection of an anomaly, the cumulative sums C_t^+ and C_t^- are reinitialized to 0.

In an improved version of CUSUM called FIR CUSUM (for Fast Initial Response) [Lucas and Crosier, 1982b], a headstart is introduced to improve the response time of the algorithm. When the process is out-of-control at the start-up, or when it is restarted after an adjustment, the standard CUSUM may be slow in detecting a shift in the process mean that is present immediately after the start-up of the adjustment. To overcome this problem, the headstart consists of setting the starting values C_t^+ and C_t^- equal to some nonzero value, typically $H/2$.

5.3.2 Choice of the parameters

The performance of the CUSUM algorithm is closely dependent on the choice of the two key parameters h and k. Two objectives must be achieved when setting these parameters. On the one hand, one must minimize the false positives. In other words, when the process is in-control, ideally, the CUSUM algorithm should not detect any change. On the other hand, any mean shift has to be detected as soon as possible. There is clearly a trade-off between these two objectives. According to [Montgomery, 2007], it is recommended to take $k = 0.5$ and $h = 4\ or\ 5$. A complete theoretical study of the performance of CUSUM is provided by Siegmund in [Siegmund, 2013]. It is based on an approximation of the ARL properties.

ARL_δ is defined as the expected number of items required by CUSUM to detect a deviation when the process has a mean deviation of δ. The approximation given by [Siegmund, 2013] is simple compared to other approaches based on approximating transitions from the in-control to the out-of-control state with a Markov chain (see [Brook and Evans, 1972]). We choose to focus on positive shifts (the problem is completely analog for negative shifts) so we consider a one-sided CUSUM where only C_i^+ is handled. In this case, Siegmund's approximation of ARL is:

$$ARL = \frac{\exp(-2(\delta - k)h') + 2(\delta - k)h' - 1}{2(\delta - k)^2}$$

where δ is the mean process shift, in the units of σ_0, and $h' = h + 1.166$. In this equation, the observed process is assumed to be normally distributed.

We first plotted in Figure 5.1 the variation of ARL_0 based on this equation. Recall that ARL_0 should be high to minimize false positives. According to this Figure, h must be taken at least equal to 3 to have an ARL_0 higher than 100, when k is close to 0.5.

The second step is to minimize ARL_δ to detect the deviation as soon as possible and to achieve a good reactivity. According to Figure 5.2, the ARL decreases when h decreases, that is why h should be small to have a better reactivity in case of a mean shift. For $h \in [3, 5]$, ARL_1 is between 5 and 11 which corresponds to a reasonable

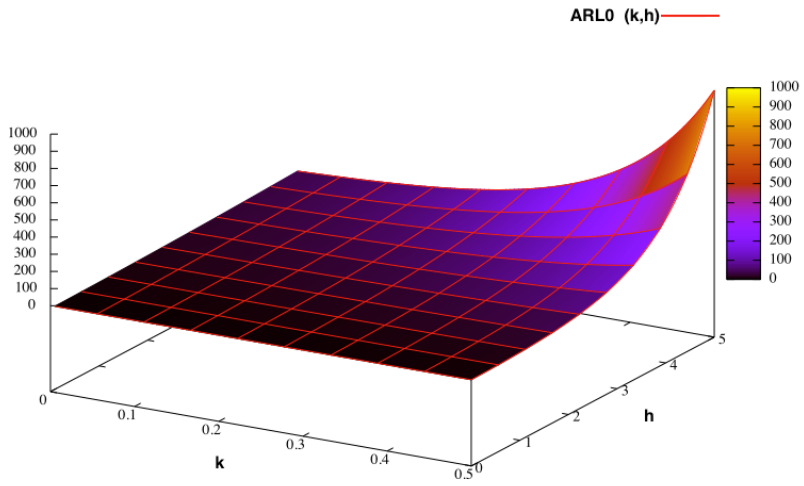


FIGURE 5.1 – $ARL_0(k, h)$: Impact of k and the control limit h on ARL_0.

response time or reactivity. This result is confirmed by Figure 5.4 which shows that CUSUM has a small ARL_δ for large shifts ($\delta \geq 1$).

In Figures 5.3 and 5.5, we focused on very small shifts detection ($\delta \in [0, 1]$). When the mean deviation δ is small, ARL_δ becomes high, close to ARL_0. One can conclude that the shift δ has to be at least equal to 0.5 to guarantee an order of magnitude of difference between ARL_0 and ARL_δ ($ARL_0 \sim 100$ and $ARL_\delta \sim 10$).

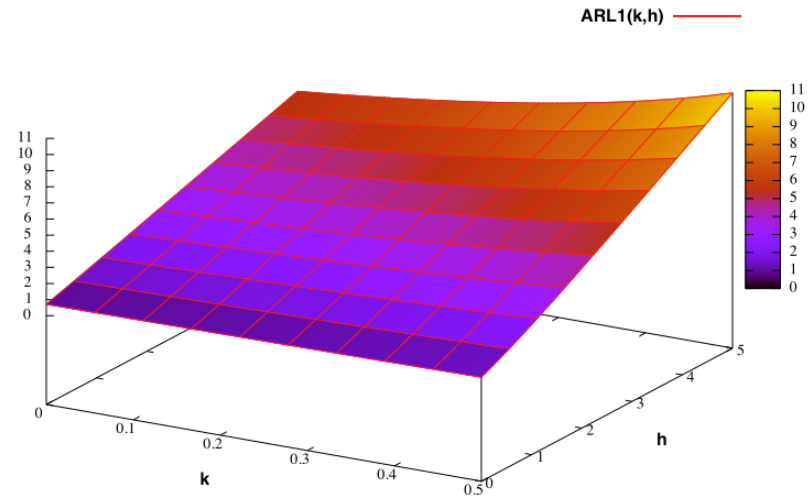


FIGURE 5.2 – $ARL_1(k, h)$: Impact of k and the control limit h on ARL_1.

The main conclusions that we can draw from this section are the following:

- The value of h must be at least 3 to have an ARL_0 greater than 100 when k is close to 0.5.
- The value of ARL decreases with the decrease of h, therefore, the value of h must be small to have a better reactivity in the case of process mean deviation.
- The deviation δ must be at least 0.5 to guarantee an order of magnitude of difference between ARL_0 and ARL_δ ($ARL_0 \sim 100$ and $ARL_\delta \sim 10$).

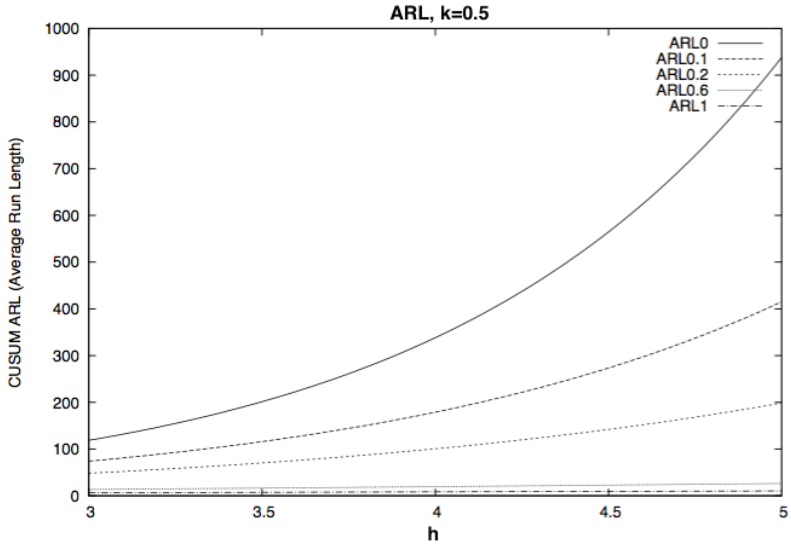


FIGURE 5.3 – Impact of the control limit h on ARL_δ.

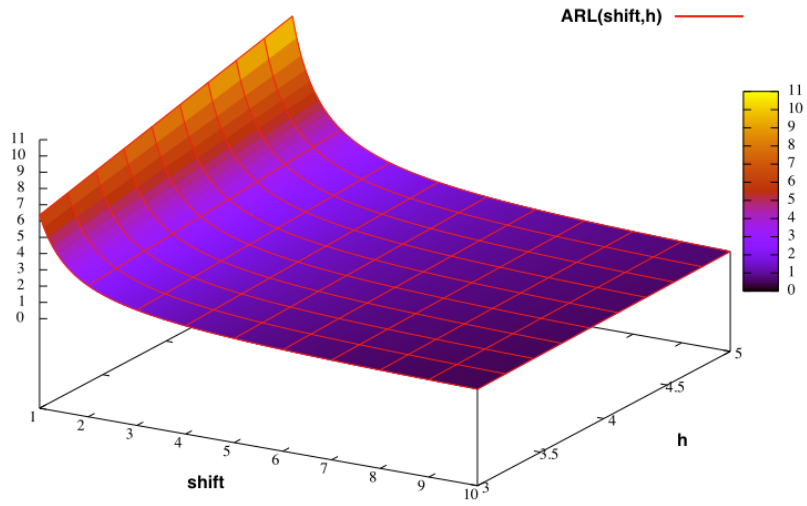


FIGURE 5.4 – Impact of the shift δ on ARL, $h \in [3, 5]$.

5.3.3 Variability of the Run Length (RL)

An important criterion used to evaluate the reactivity of a control chart algorithm is the ARL. We recall that it represents the expected number of observations needed before an out-of-control alarm is detected. ARL has two interpretations. ARL_0 is defined as the average number of in-control data needed by the control chart algorithm to signal a false alarm. ARL_δ is the average number of out-of-control data required to detect the error after a process mean changed. ARL_0 has to be as large as possible, and ARL_δ has to be small in order to detect the shift quickly. The Run Length is a random variable. To our knowledge, only its average was theoretically studied. No theoretical results about the variability of Run Length are provided in the literature. In this section, we address this problem using simulations.

We report in Table 5.1 the average and the standard deviation of the RL, respec-

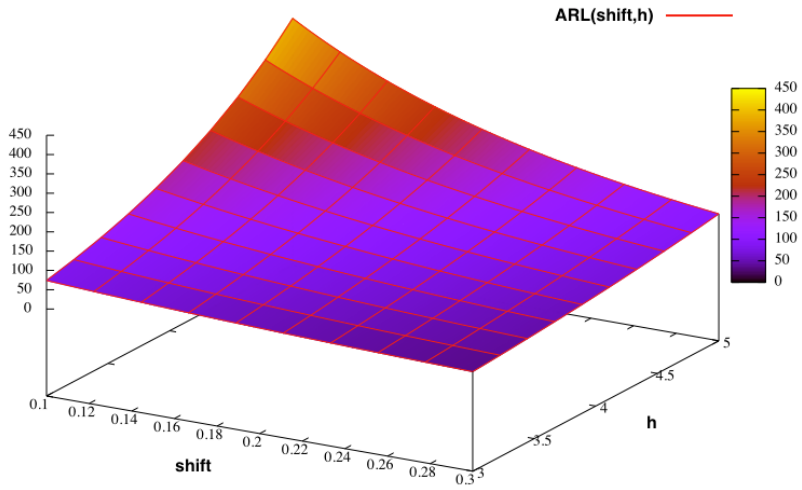


FIGURE 5.5 – Impact of small shifts δ on ARL, $h \in [3, 5]$.

tively ARL and $SDRL$, for different values of shift δ. According to the recommendation of [Montgomery, 2007], we choose $k = 0.5$ and $h = 4$ in order to have a good ARL for a shift of 1σ in the process mean.

In order to simulate ARL_0, a sample of size 1000 is generated under an in-control situation. CUSUM chart is then applied to the samples until an out-of-control signal is triggered. The number of observations when the signal is triggered is the in-control Run Length RL_0. We repeat this simulation 150 times so we can get the average value: ARL_0. The considered process follows the standard normal distribution. To simulate the ARL_δ, we performed the same experiments with an injected shift of δ in the process mean. We can notice that the average and the standard deviation of the RL decrease with the increase of the shift δ. Moreover, the false positives detected by CUSUM closely depend on the execution, as they are calculated using RL_0 which has a high standard deviation compared to its average ARL_0. Therefore the Run Length to obtain a false positive can be significantly lower than ARL_0. One can conclude that for a single execution, the CUSUM can lead quickly to a false positive.

TABLE 5.1 – Simulated Run Lengths (RL), for different shift sizes.

	Shift δ					
	0	**0.25**	**0.5**	**1**	**2**	**3**
Theoretical ARL	370	74.2	26.6	8.34	3.34	2.19
Simulated ARL	328	73.59	25.82	8.02	3.38	2.74
SDRL	220	8.38	1.92	1.77	1.41	1.31

5.4 Enhancing the reactivity of CUSUM algorithm

Recall that to detect positive mean shifts, CUSUM is based on the following cumulative statistic:

$$C_t^+ = max[0, s_t - \mu_0 - K + C_{t-1}^+]$$

In the standard version of CUSUM, the deviation from the expected average is declared after performing enough iteration to deeply impact C_t. However, it is not possible to know the exact start time of the small shift. In fact, the change detection occurs after real change start time and no estimation of this latter is provided in the literature. Moreover, the observed process is considered as in-control (end of the deviation) when C_t is less than the detection threshold H. As C_t is a cumulative sum, it sometimes takes a long time (many iterations) to achieve this condition. Thus, the end of deviation is declared a long time after the real return to the standard behavior.

To overcome these problems, we propose the following improvements:

- Add an estimation of the start time of the change.
- Improve the precision of the end time of the change.

5.4.1 Detecting the anomaly start time

The first improvement of CUSUM is related to the start time of the anomaly. In the literature, the change is simply declared when it is detected. Let us take $K = \sigma_0/2$, and s_t a process normally distributed: $S \sim \mathcal{N}(\mu_0, \sigma_0^2)$.

When the process s_t is in-control, $(s_t - \mu_0 - K)$ has the same probability of being positive or negative for symmetry reasons. When s_t has a mean positive shift (of σ_0 as an example), $(s_t - \mu_0 - K)$ becomes very likely to be positive. Therefore, C_t is very likely to be increasing ($C_t > C_{t-1}$). This is the key idea behind our improvement. When the process is out-of-control, the value of C_t is very likely to increase as long as the deviations persist.

When an error is detected ($C_t > H$), the start time of this anomaly can be estimated by the moment where C_t became strictly increasing. This moment can be inferred even if it is former to the detection of the error. For this purpose, we introduce a Start Time counter (ST), that we update each time we calculate C_t. If an error is declared at time t, its start time is estimated as: error start time = $t - ST + 1$.

Initially, ST is set to 0, then, it is updated as follows:

- $ST \leftarrow ST + 1$ if the value of C_t increases.
- $ST \leftarrow ST - 1$ if the value of C_t decreases.
- ST is reset to 0 if the end of the error is detected.

In the case of an in-control process, ST has a random distribution with a null mean. The counter ST is used to estimate the size of each anomaly, in other words, the process out-of-control duration.

5.4.2 Detecting the anomaly end time

As far as we know, the real end time estimation of the anomaly for CUSUM algorithm was not addressed before. In the standard version of CUSUM, the end of the anomaly is declared when C_t becomes lower than the threshold H. Being a cumulated sum, C_t needs many steps (or iterations) to attain its normal values ($< H$) after the end of the error. To improve the reactivity of the CUSUM algorithm, we introduce an End Time counter (ET), to be able to detect the end of the anomaly quickly.

The key idea of this improvement is that when C_t becomes constant or decreasing, the current deviation is very likely to be stopped. The condition $C_t^- \leq C_{t-1}^-$ is always achieved before $C_t^- < H$ as in case of error $C_{t-1}^- > H$. The counter ET is updated each time we calculate C_t. It depicts the number of successive decreases of C_t.

Initially, ET is set to 0, then, it is updated as follows:

- $ET \leftarrow ET + 1$ if C_t decreases.
- $ET \leftarrow 0$ otherwise.

The end of the anomaly is declared when ET exceeds a given threshold ET_0. Just like σ_0 and μ_0, this latter is inferred from the training window. It is the average number of successive decreases in the training window (when the process is in-control).

The detailed pseudo code of the anomaly detection using the improved CUSUM algorithm is given by Algorithm 8.

5.5 Detecting mean change

5.5.1 Efficiency metrics

In this section, we evaluate the performance of CUSUM algorithm for the anomalies detection. In information retrieval domain, the performance metrics used to evaluate the performance of a change detector are precision, recall and specificity. These metrics are based on the True Positives (TP), False Positives (FP), True Negatives (TN) and False Negatives (FN). The errors committed by CUSUM can either be false positives (the process is considered as out-of-control while it is not true) or false negatives (no alarm is signaled whereas the process is out-of-control).

- True Positive *(TP)*: it represents the state where an actual data point is an error, and it is detected by the algorithm as an error.
- False Negative *(FN)*: it represents the state where an actual data point is an error, and it is not detected by the algorithm as an error.
- False Positive *(FP)*: also called *false alarm*, represents the state where an actual point is not an error, but it is detected by the algorithm as an error.
- True Negative *(TN)*: also called *correct rejection*, represents the state where an actual point is not an error, and it is not detected by the algorithm as an error.

Algorithm 8. Anomaly detection with the enhanced version of CUSUM algorithm

```
1: procedure AnomalyDetection(K = σ0/2, H = 4σ0)
2:     C0 ← 0
3:     ST ← 0
4:     ET ← 0
5:     errorInProgress ← false
6:     for each incoming item st, t > 0 do
7:         Ct= max[0, st − μ0 − K + Ct−1]
8:         if (Ct > Ct−1) then
9:             ST ← ST + 1
10:            ET ← 0
11:            if (Ct > H) then
12:                errorStartTime ← t − ST + 1
13:                return errorStartTime
14:                errorInProgress ← true
15:            end if
16:        end if
17:        if (Ct < Ct−1) then
18:            ST ← ST − 1
19:            ET ← ET + 1
20:            if (errorInProgress == true) then
21:                if (ET > ET0) then
22:                    errorEndTime ← t − 1
23:                    return errorEndTime
24:                    errorInProgress ← false
25:                    Ct ← 0
26:                    ST ← 0
27:                end if
28:            end if
29:        end if
30:    end for
31: end procedure
```

Precision metric is the proportion of true alarms compared to all the alarms raised by CUSUM. *Recall* metric, also called *sensitivity*, depicts the true positive rate: the probability that CUSUM algorithm identifies a truly erroneous point. *Specificity* metric represents the true negative rate: the proportion of points considered truly as non-erroneous by CUSUM compared to the total number of non-erroneous points present in the dataset.

$$Precision = \frac{TP}{TP + FP}$$

$$Recall = \frac{TP}{TP + FN}$$

$$Specificity = \frac{TN}{TN + FP}$$

5.5.2 Experimentations

The objective of this section is to validate our proposed improvements of CUSUM algorithm to detect a negative shift of the mean among normally distributed simulated data. Let us take a process $(S_t)_{t\geq 0}$ that follows the standard normal distribution: $S \sim \mathcal{N}(0,\, 1)$. We first considered 1000 observations of S, then we injected at random moments 10 deviations (also called errors). The purpose of the experiments is to apply CUSUM to detect these deviations and to use our improvements to estimate the start time and the end time of each detected deviation. The errors have a random length taken in $[1, 50]$. The cumulative errors length equals 239 points or observations. As the targeted change is a mean shift, for each error, we replaced the original points by new observations issued from a shifted process $S' \sim \mathcal{N}(-1,\, 1)$. Notice that the variance of the process remains unchanged.

As only negative shift is considered in this section, we only focus on the cumulative parameter C_t^-, that we simply denote by C in the following parts.

$$C_t^- = max[0, \mu_0 - s_t - K + C_{t-1}^-]$$

k is set to 0.5 and the threshold h is taken equal to 4, according to the recommendations given in Section 5.3.2.

The variation of C over time, together with the detection threshold H are plotted in Figure 5.6. We can notice that C is very variable over time and presents several peaks that are mainly caused by the injected errors. CUSUM reports errors each time the value of C exceeds the control limit H. We obtained a total number of 8 alarms corresponding to the 8 detected errors. Two errors are missed because they have very small durations.

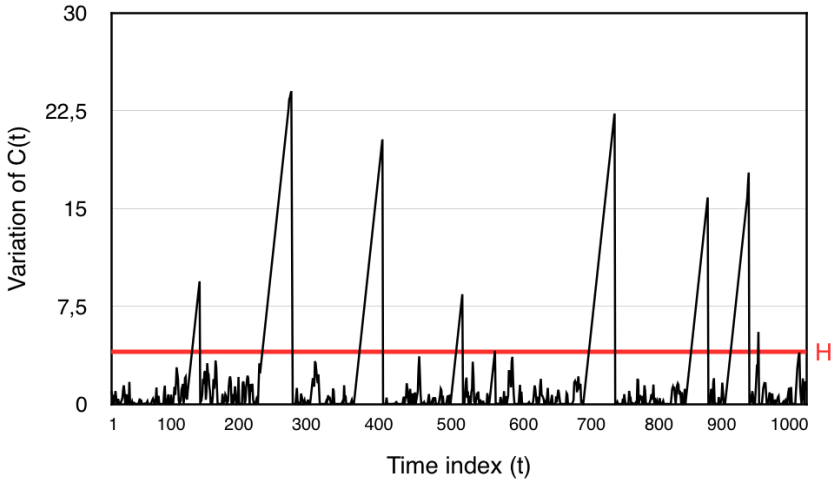


FIGURE 5.6 – Variation of C_t over time.

Figure 5.7 depicts the variation of the counter ST over time. Recall that ST is used to estimate the start time of the error. It is incremented by one with the increase of C and decremented C decreases. Figure 5.7 shows that when the process is in-control, the value of ST has a null average and very small standard deviation. The presence of an error induces a notable increase of ST.

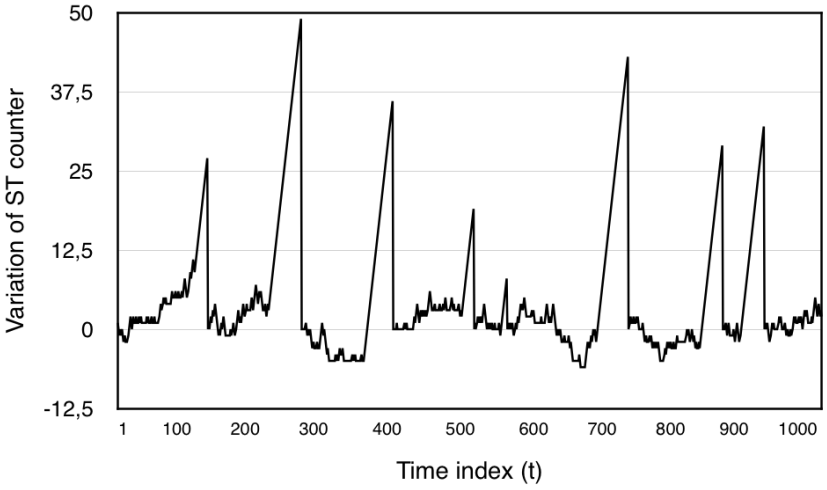


FIGURE 5.7 – Variation of ST counter over time.

The variation of the counter ET over time is shown in Figure 5.8. ET counter enables to estimate the end time of the error. It counts the number of successive decreases of C. We can see that ET has small variations with a mean of 0.22 and a standard deviation of 0.57. The end of the error is declared when ET exceeds its average value ET_0 calculated in the training window. In our case, ET_0 equals to 0.25.

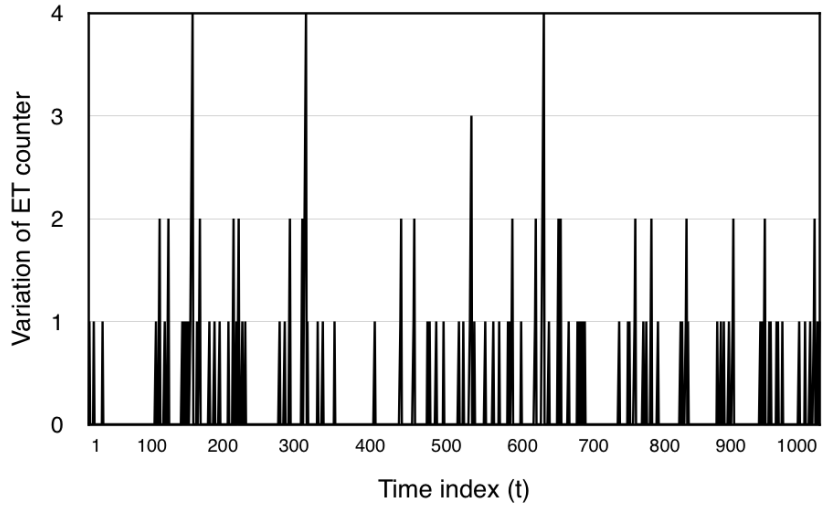


FIGURE 5.8 – Variation of ET counter over time.

The evaluation of the proposed improvements is given in Table 5.2. The 1000 considered points of the process are first classified into actual error and actual not error. Then we add a second classification according to the detection results of the improved CUSUM. The same experiments are performed with the standard version of CUSUM and FIR CUSUM. Recall that FIR CUSUM is an improved version of CUSUM. With the classic version of CUSUM algorithm, the values C_t^+ and C_t^- are reset to zero after the detection of a change. The objective of FIR CUSUM is to improve the performance of CUSUM by setting the values C_t^+ and C_t^- equal to some nonzero value, typically $H/2$. The obtained results are presented in Table 5.2. One can notice that our proposed algorithm outperforms both standard and FIR CUSUM. It significantly decreased both false negatives and false positives.

TABLE 5.2 – Obtained results for mean change detection.

CUSUM version	Actual \ Detected as	anomaly	not anomaly
Standard	anomaly	189	50
	not anomaly	203	558
FIR CUSUM	anomaly	174	65
	not anomaly	196	565
Improved	anomaly	**221**	**18**
	not anomaly	**23**	**738**

The three efficiency metrics (precision, recall, and specificity) of the standard CUSUM, FIR CUSUM and the improved version of CUSUM are given in Table 5.3. All these metrics are enhanced using our improved version of CUSUM. The proposed improvements give very good results as the three efficiency metrics are above 0.9.

TABLE 5.3 – Performance metrics of CUSUM.

CUSUM version	Precision	Recall	Specificity
Standard	0.48	0.79	0.73
FIR CUSUM	0.47	0.72	0.74
Improved	**0.90**	**0.92**	**0.96**

5.6 Application to stuck-at error: Detecting variation change

In this section, we apply the improved CUSUM algorithm on a real data stream issued from water flowmeters, to detect the so-called stuck-at errors. During the data analysis process, the conclusions and decisions are based on the data. If the data are dirty, this will lead to defective and faulty results. Improving the data quality is thus inevitable to obtain reliable results. One of the data quality measures is the *accuracy*. It represents the difference between the observation's value and the true value which the sensor aims to represent. Due to the instrumental, physical and human limitations, malfunction and miscalibration of the sensor, the observations values of the sensor can deviate significantly from the true ones. These deviated values are called faults.

The stuck-at error, also called CONSTANT in [Ramanathan et al., 2006, Sharma et al., 2010], is a type of sensor errors. A stuck-at x error occurs when the sensor is stuck on an incorrect value x. The low battery of the sensor, a dead sensor, or the malfunction of the sensor, may cause this error. During such a situation, a set of successive observations will have the same value $x \pm \epsilon$. The error may last a long time, and the sensor may or not return to its normal behavior. [Sharma et al., 2010]

showed that CONSTANT errors are present in the sensors data of the INTEL Lab and in NAMOS data set. This kind of error concerns about 20% of their data.

The variance is the characteristic to be modeled in order to detect this type of error, during which, the variance of the data drops significantly [Ni et al., 2009]. However, the variance modeling requires the use of a temporal window with a specific size, which is not a trivial task. If the size of the window is very large, the error can be missed. On the contrary, if the size of the window is very small, this can engender false positive detections. Nair *et al.* [Nair, 2009] calculates the absolute value of differences between successive values to detect stuck-at errors. Thus, at each instant t, the difference d between the value at the instant t and that at the instant $t-1$ is calculated. If d is less than a certain threshold for a certain number of successive values, the data at these instants will be considered as stuck-at errors. The determination of the threshold and the number of successive points from which the data are considered stuck-at represent the main challenges of this method.

In this section, we propose to use CUSUM algorithm to detect the stuck-at errors, and we use the improved version of the algorithm presented in Section 5.4. As the change concerns the variance of the observed process, we choose to apply CUSUM on the variations v_t of the observed values: $v_t = |s_t - s_{t-1}|$. $(V_t)_{t\geq 0}$ is a positive time series with an average of μ_0 and a standard deviation σ_0 during the training window. As the stuck-at error engenders a decrease of μ_0, we only focus here on the cumulative statistic C_t^-, for a one-sided CUSUM.

The dataset duration considered in this section is of 10 days in January 2014, with a total number of 960 observations. To inject a stuck-at error in the time series, we chose random instant t and we replaced s_t by a random variable uniformly distributed in $[s_t - \mu_0 + \sigma_0, s_t + \mu_0 - \sigma_0]$, for a random number of successive values beginning from the instant t. Hence the mean of the process $v_t = |s_t - s_{t-1}|$ drops from μ_0 to $\mu_0 - \sigma_0$. The shift that we want to detect is about $-\sigma_0$. We repeated this mechanism 10 times to inject 10 errors. The errors have a random length taken in $[1, 50]$. We obtained a total number of 263 injected erroneous points.

Figure 5.9 shows the considered dataset s_t with the injected errors. The variations of s_t denoted as $v_t = |s_t - s_{t-1}|$ are plotted in Figure 5.10. One can easily notice a periodicity in the water consumption with a difference between the working and not working days. Moreover, during the injected errors, the variations of s_t denoted as $v_t = |s_t - s_{t-1}|$ drop significantly. Recall that we performed CUSUM on v_t.

The variation of C_t over time is given by Figure 5.11. 8 alarms were reported by CUSUM when C_t exceeds the threshold H. They correspond to real injected errors. CUSUM missed two errors as they have very small durations (of only 1 and 4 points), compared to ARL_1. In fact, according to Section 5.3.2, ARL_1 equals 8.38 (when $k = 0.5$, $\delta = \sigma$ and $h = 4$). It means that we need in average 8 observations to could detect this change.

Using the same efficiency metric presented in Section 5.5.1, we recap in Tables 5.4 and 5.5 the obtained results of the variation change detection, after applying the improved and the standard version of CUSUM algorithm.

Just like in the previous subsection (Section 5.5.2), we checked, based on these values that the proposed improvements enhance the three efficiency metrics: the

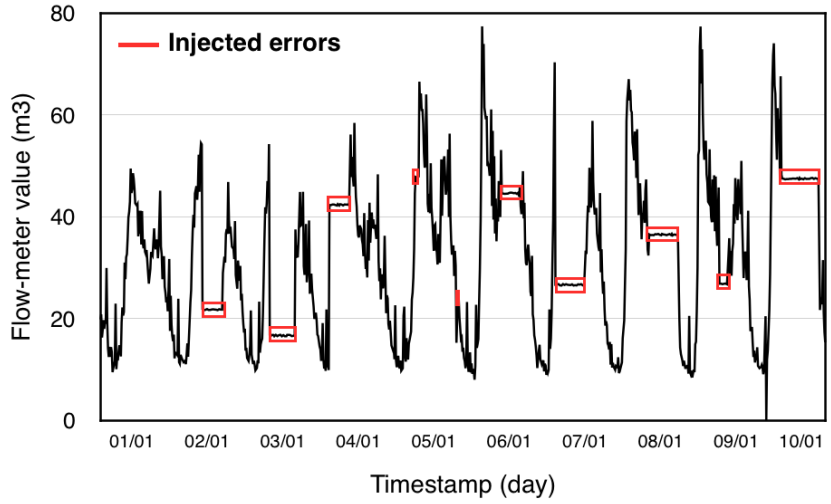


FIGURE 5.9 – Injection of variation change errors.

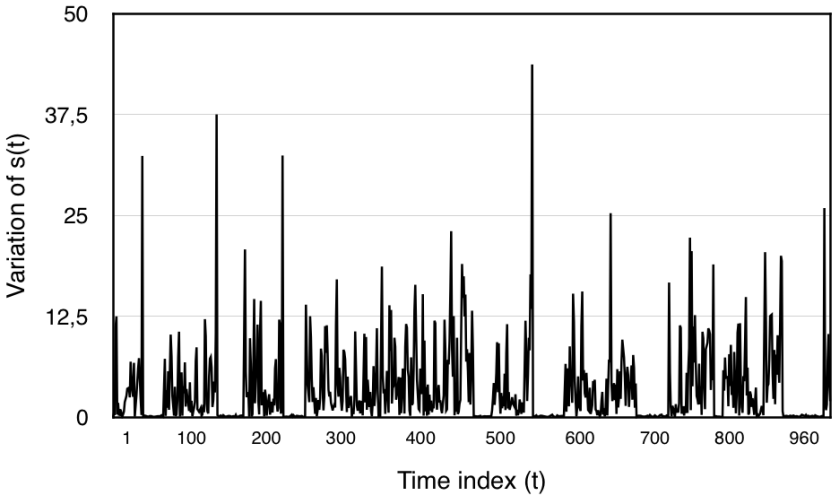


FIGURE 5.10 – Variation v_t of s_t after the injection of errors.

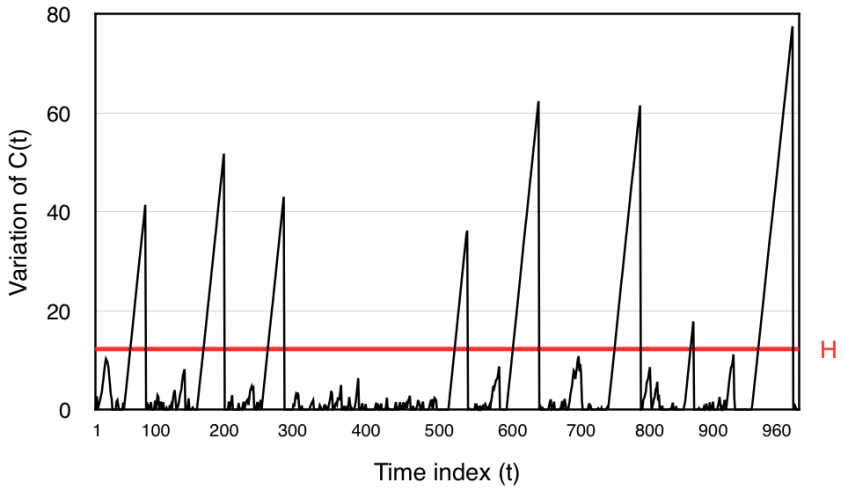


FIGURE 5.11 – Variation of C_t over time.

TABLE 5.4 – Obtained results for stuck-at errors detection.

CUSUM version	Actual \ Detected as	stuck-at error	not stuck-at error
Standard	stuck-at error	134	129
	not stuck-at error	121	576
Improved	stuck-at error	**255**	**8**
	not stuck-at error	**68**	**629**

precision, the recall and the specificity of the CUSUM algorithm (see Table 5.5). Moreover, these results show that the CUSUM algorithm performs good results even if the considered data set does not verify the normal distribution.

TABLE 5.5 – Performance metrics of CUSUM.

CUSUM version	Precision	Recall	Specificity
Standard	0.52	0.50	0.82
Improved	**0.78**	**0.97**	**0.90**

5.7 Conclusion

The performance of the CUSUM algorithm is closely dependent on the choice of two key parameters: h and k. Two objectives must be achieved when defining these two parameters. On the one hand, it is necessary to minimize the false positives: when the process is under control, CUSUM algorithm should not detect a change. On the other hand, any deviation from the mean should be detected as soon as possible. In this chapter, we studied at first the choice of the two parameters h and k, then, we studied the Run Length (RL_δ) variability for a deviation $\delta \in [0, 3]$.

Secondly, we proposed an efficient method for the estimation of the start time and end time of the process mean deviation. In fact, the deviation is declared after performing a number of iterations sufficient to have a significant impact on C_t. Thus, the deviation is declared after the exact start time of the actual change. In the standard version of CUSUM algorithm, it is not possible to know the exact start time of the deviation, and no estimation of this time is provided in the literature. Furthermore, the observed process is considered under control (end of the deviation) when C_t is below threshold H. Since C_t is a cumulative sum, CUSUM needs many iterations to reach this condition. Thus, the end of the deviation is declared long after the actual return of the process to the normal behavior.

To improve the reactivity of CUSUM algorithm, we proposed these two improvements:

- Add an estimation of the start time of the deviation.
- Improve the accuracy of the end time estimation of the deviation.

We compared the performance of the proposed new version of CUSUM algorithm to that of the standard version of CUSUM and another version called FIR CUSUM. The results of the experiments showed that the new version outperforms the other two versions of the algorithm.

Thereafter, we adapted the CUSUM algorithm to detect the so-called stuck-at errors. A stuck-at x error occurs when a set of successive values have the same value $x \pm \epsilon$. We tested the performance of the standard version and the proposed new version of the CUSUM algorithm to detect the stuck-at errors. The results showed that the enhanced version of CUSUM algorithm improves the three efficiency metrics precision, recall and specificity of the detection.

Notice that, the detected stuck-at errors may be due to a sensor defect or to a real phenomenon. The intervention of the network explorer and his knowledge of the environment make the differentiation possible. Several factors about the environment are to be evaluated, in our use case, the profile of the sector (residential, agriculture, industrial, etc.) and the length of the stuck-at series.

Chapter

6 Spatial outliers detection with Moran Scatterplot

Contents

6.1 Introduction **107**
6.2 Motivation **108**
6.3 Dataset description and problem definition **110**
6.4 Spatial outliers detection with an improved Moran scatterplot **111**
6.4.1 Moran scatterplot 111
6.4.2 Improvement of Moran scatterplot using Gower's coefficient 113
6.5 Enhancing resources distribution in Velib system **115**
6.6 Conclusion **120**

The scientific contribution and the obtained results presented in this chapter have been published in our paper: **Rayane El Sibai**, Yousra Chabchoub and Christine Fricker. *"Using spatial outliers detection to assess balancing mechanisms in bike sharing systems"*. In the 32^{th} IEEE International Conference on Advanced Information Networking and Applications (AINA), IEEE, 2018.

6.1 Introduction

Anomalies, also called outliers, are deviant observations compared to a standard behavior. The identification of outliers has many practical applications in many areas, such as intrusion detection, fraud detection, and fault detection. Outliers detection is also an important task in the data analysis process, it aims to detect the anomalies and leads to the identification of unusual phenomena and the discovery of new knowledge concerning the monitored environment.

Our goal in this chapter is the anomalies detection in a spatial context, where an object is considered as an outlier if the values of its non-spatial attributes are significantly different from those of other objects in its surrounding. Spatial outliers detection is useful in many applications, such as abnormal road traffic patterns detection, identification of outbreaks of disease, detection of tornadoes and hurricanes, and identification of soil pollution in urban areas.

Our use case in this chapter is the Parisian bike sharing system (Velib). Our work is motivated by the problem of heterogeneity in bikes stations. In fact, some stations are often almost empty, without enough available bikes or almost full causing users dissatisfaction. In this context, we define outliers as stations with a lack of resources while the other stations in the neighborhood are globally balanced. First, we use an adapted version of Moran scatterplot to explore and characterize the neighborhood of such stations. The results show a local heterogeneity in Velib station: in a small area, bikes availability is often very variable, depending on the station. This local heterogeneity motivates us to propose a new incentive method which encourages users to improve bikes' distribution among the Velib stations. This mechanism is based on a local small change in users trips. In this ecological regulation, users are redirected to another station in the neighborhood of their source or destination to locally reduce stations heterogeneity.

This chapter is organized as follows. We highlight in Section 6.2 the problem we are going to deal with in this chapter. We present in Section 6.3 the dataset used in this work and we discuss the main problem of Velib system. In Section 6.4, we describe the so-called Moran scatterplot technique and the proposed adaption to Velib context. We also detail in this section the experiments carried out to illustrate the heterogeneity of Velib system. In Section 6.5, we present and validate our new solution to balance the Velib system and to improve bikes distribution among the stations. We end the chapter with a conclusion.

6.2 Motivation

Outliers are defined as a set of observations that are inconsistent with the remainder observations. Outliers identification has practical applications in many areas, such as intrusion detection, fraud detection, fault detection and medical informatics [Chandola et al., 2009]. Outliers detection is also an important task in the data analysis process. It aims to detect abnormal patterns and leads to the identification of unusual phenomena, and new knowledge about the monitored environment. To isolate outliers it is necessary to first characterize the normal observations, which can be provided by the past values of the same object or by the current values issued from other objects in the neighborhood. In this latter case, the outlier is said spatial. In a spatial context, each data is defined with two categories of attributes: spatial attributes and non-spatial attributes. Spatial attributes include the shape, position, and other topological characteristics of the sensor, and they are used to define the neighborhood of the spatial object. Non-spatial attributes include the ID, manufacturer, age, and sensor measure (called behavioral attribute). A spatial outlier represents a local instability and is only compared to the surrounding dataset [Shekhar et al., 2011]. This is based on the rule: *"Everything is related*

to everything else, but nearby things are more related than distant things" [Tobler, 1970]. Spatial outliers detection serve in many applications, such as the detection of abnormal highway traffic patterns [Shekhar et al., 2001], the identification of disease outbreaks [Wong et al., 2002], the detection of tornadoes and hurricanes [Lu and Liang, 2004] and the identification of urban soils pollution [Zhang et al., 2008].

Several algorithms have been developed to detect the outliers in a spatial context. These algorithms can be classified into two categories: graphical-based algorithms, and quantitative-based algorithms. Graphical-based algorithms use visualization. They present, for each spatial point, the distribution of its neighbors and identify outliers as points in specific regions. This category includes Variogram Cloud, Pocket Plot, Scatterplot, and Moran scatterplot methods. Quantitative-based algorithms perform statistical tests to distinguish the outliers from the rest of the data. These methods include z algorithm, iterative r, iterative z and median algorithm.

Scatterplot represents the data in a two-dimensional space where the X-axis represents the values of the non-spatial attribute (the observable) of each object and the Y-axis represents the mean value of the observable of the neighbors of this object. A regression line is used to identify outliers points [Haining, 1993]. Variogram Cloud [Haslett et al., 1991] is a scatterplot between the spatial distance (X-axis) and the difference of the observable values (Y-axis) for each pair of points in the data. Outliers points are identified as pairs of points having a small spatial distance and a big difference for the observables measurements.

The Z statistic approach [Shekhar et al., 2003] is one of the most known quantitative-based algorithms for spatial outliers detection. For each spatial object x, S_x denotes the difference between the attribute value of x and the average attribute value of its spatial neighbors. Spatial outliers are simply identified using a threshold based on μ_s and σ_s which respectively represent the mean and the standard deviation of the attribute value of S over all the spatial objects.

In [Lu et al., 2003], the authors propose two iterative algorithms (iterative r and iterative z) for the detection of spatial outliers. These algorithms detect the outliers on several iterations. Each iteration detects a single outlier and modifies its value in order to reduce its negative impact on its neighbors in the next iteration

We apply in this chapter the spatial outliers problem to a particular case study: the evaluation of a balancing mechanism in Bike Sharing Systems (BSS). Nowadays, public authorities are more and more encouraging this ecological mean of transport by expanding the BSS to the suburbs and building new bike paths. Since its launch in 2007, Velib (the Bike Sharing System -BSS- in Paris) has emerged in the Parisian landscape and has been a model for similar systems in many international cities. Velib provides a significant proportion of people travels as it daily ensures about $110,000$ trips. It involves about 1800 stations with an average distance of 300 meters.

A major problem in the Velib system and in BSS, in general, is the problem of empty stations and full stations caused by the asymmetric attendance to the stations. According to the annual satisfaction survey of Velib, only 50% of users are satisfied with the availability of bikes and docks in the stations [vel, 2014]. Despite the performed regulation (moved bikes using trucks), users often find themselves in front of stations that are totally full or empty.

In most cities, operators provide open access to real-time status reports on their

bike stations. Several studies show the interest of using these data (Froelich *et al.* [Froehlich et al., 2009] and Borgnat *et al.* [Borgnat et al., 2011], Vogel and Mattfeld [Vogel et al., 2011]). Their main objective is to understand and characterize the behavior of the users in order to help in designing and planning policy in urban transportation. Among these studies, one can cite the partition of the BSS stations into several classes using different clustering algorithms (see [Chabchoub and C., 2014] and [Etienne and Latifa, 2014] for more details). Other studies, performed a classification of the flows of trips as analysis of the trips in the Velo's system in Lyon proposed by Borgnat *et al.* in [Borgnat et al., 2011].

6.3 Dataset description and problem definition

In order to promote innovation and collaboration with scientists, different kinds of data relative to the Velib system are "Open Data" available for the research community. We performed all the experiments presented in this chapter from this dataset which has been presented in the introduction of the report (Velib dataset).

We recall that this dataset consists of two types of data. First, we have the static data describing the Velib stations. They consist of spatial attributes: the geo-coordinates of the station (latitude and longitude), and non-spatial attributes: Id of the station and its capacity (total number of docks). Then, the dynamic data are of two kinds: First, the number of bikes present in each station for each timestamp t are provided in real-time. This parameter is varying during the day and is closely dependent on users activity. Second, Velib users data trips are also available (one file for the trips during a month). A trip is characterized by a departure and arrival timestamp, and a departure and arrival station. The analysis of several months of trips showed a very strong periodicity: trips can be divided into two main categories: the working days and the weekends. Two days of the same category are very similar. We focus in this chapter on the working days and we choose to analyze 24 hours trips: trips that took place on Thursday, October, the 31^{th}, 2013. This duration includes 121.709 trips, involving 1226 Velib stations. 1.03% of the trips are related to maintenance (bikes taken for repair) and 1.48% are trips of regulation (bikes moved by trucks).

According to many research studies ([Chabchoub and Fricker, 2014], [Fricker and Gast, 2014] and [Etienne and Latifa, 2014]), the Velib system has some weaknesses caused by the strong attractivity of some stations that can be explained by their location near a railway station or a monument or a business area. Such stations are very often almost empty (no available bike) or completely full (no available dock to put a bike). Despite the performed regulation (bikes moved by trucks), the system is still unbalanced. This unbalanced distribution of bikes among the different stations causes users dissatisfaction.The unbalanced stations are referred to as *problematic* stations. More precisely, we introduce the following definition: a station is said problematic at a timestamp t if its *occupancy rate* is under 10% or more than 90%.

The *occupancy rate* of the station, at a timestamp t, is defined as follows:

$$occupancy\ rate\ _t = \frac{Number\ of\ bikes\ present\ at\ t}{Capacity\ of\ the\ station} \times 100\%$$

6.4 Spatial outliers detection with an improved Moran scatterplot

The objective of this section is to estimate the number of isolated problematic stations at a given timestamp t, which motivates the incentive method detailed in the following section. A good understanding of the current use of the Velib' system and the real needs of the users is mandatory to improve the performance of this system and to plan its future expansion and evolution. An isolated problematic station satisfies both following conditions: First, it is almost empty or almost full at timestamp t. Second, its occupancy rate is significantly different from the average occupancy of the neighboring stations at the same timestamp t. Thus the isolated problematic stations are among the spatial outliers. In this section, we consider the system at a fixed timestamp t. In order to detect spatial outliers, we opted to use Moran scatterplot [Anselin, 1993] that we adapted to the specificities of our context.

6.4.1 Moran scatterplot

Moran scatterplot [Anselin, 1993] illustrates the similarity between an observed value and its neighboring observations. It measures the global spatial autocorrelation over a geographical area, the well-known *Moran's I*. Let us denote by $Z = \{z_i : 1 \leq i \leq n\}$ the set of the different values of the considered observable at a fixed given time t, in n different locations. For each location, the neighborhood is defined based on the geographical distance. Moran scatterplot visualizes the relationship between the values z_i and their neighborhood average $W_i.Z$, where W is a weight matrix that defines a local neighborhood around each location. The observations Z (x-axis) and $W.Z$ (y-axis) are represented by their standardized values.

Moran scatterplot contains four quadrants, corresponding to four types of spatial correlation. The upper-right and lower-left quadrants consist of the locations with positive spatial correlation: association between similar values. In the upper-right quadrant, the high values are surrounded by high neighbors values, while in the lower-left quadrant, the low values are surrounded by low neighbors values. The upper-left and lower-right quadrants incorporate the locations with negative spatial correlation: association between dissimilar values. The upper-left quadrant contains low values surrounded by high neighbors values, while the lower-right quadrant contains high values surrounded by low neighbors values. The objects located in these two quadrants are considered as spatial outliers and can be identified by the statistical test function:

$$Z_i \times \sum_j w_{ij} Z_j < 0$$

W is the contiguity matrix of weights. It indicates the spatial relationship between every couple of objects. W is also called the row-normalized neighborhood matrix. It is based on a threshold d of the geographical distance: i and j are considered as neighbors if and only if $0 \leq d_{ij} \leq d$, where d_{ij} is the distance between i

and j. Moreover, all the neighbors of i are equivalent and have the same impact on the calculation of the neighborhood average $W_i.Z$.

Thus, the contiguity matrix W is given by:

$$w_{ij} = \begin{cases} \frac{1}{Number\ of\ neighbors\ of\ i}, & \text{if } 0 \leq d_{ij} \leq d \\ 0, & \text{otherwise.} \end{cases} \tag{6.1}$$

To apply Moran scatterplot to the context of Velib, one has to estimate the crucial parameter d, which represents the highest distance between two neighboring Velib stations. The choice of d has to achieve the following trade-off: On the one hand, this distance has to be small enough to let the users slightly change their trips at a local scale, and on the other hand, it has to be high to make sure that most stations have a reasonable number of neighboring stations. Velib stations are generally close to each other and concentrated in the center of Paris and near attractive locations whereas they are distant in the suburbs.

To address this problem, we plotted in Figure 6.1 the distribution of the number of neighbors for all the Velib stations. We tested different values for the threshold distance d (300, 400 and 500 meters). According to Figure 6.1, a distance of 400 meters is reasonable as, in this case, a given Velib station has on average about 5 neighboring stations. Moreover, with $d = 400$, only 4.4% of the stations do not have any neighboring station.

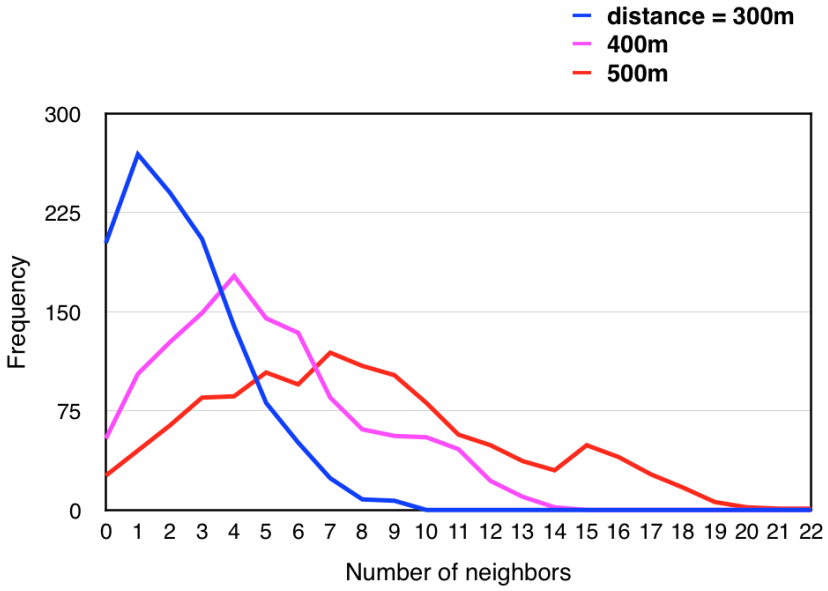


FIGURE 6.1 – Distribution of the number of neighbors.

However, when detecting spatial outliers, the assumption that all the neighbors have the same impact on the neighborhood average may lead to missing some true spatial outliers. In the dataset described in Section 6.3, there are 1226 stations. As plotted in Figure 6.2, the capacity of the stations is highly variable between 8 and 114 bikes, with an average of about 31 bikes. As Velib stations have different capacities, we defined the occupancy rate in order to compare normalized bikes availability in these stations. The key idea is that two neighboring stations should have almost

the same occupancy rate if they have similar capacities. That is why the capacity of the station has to be taken into account when calculating neighborhood occupancy average.

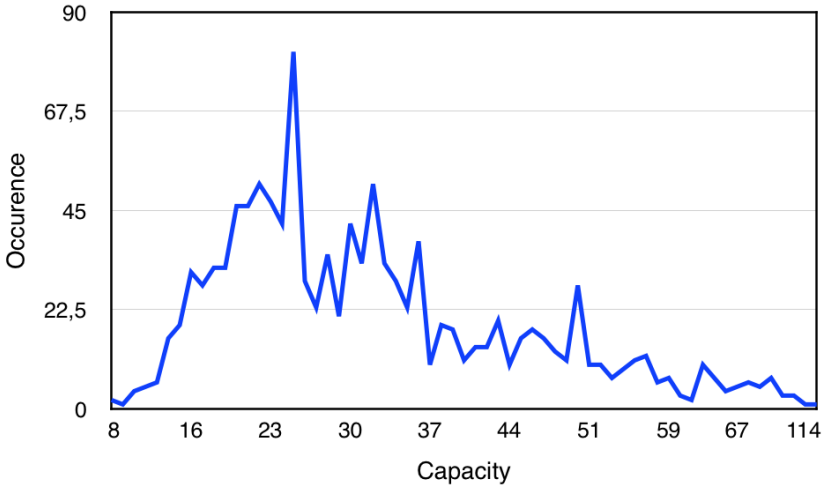


FIGURE 6.2 – Distribution stations capacity.

6.4.2 Improvement of Moran scatterplot using Gower's coefficient

We will replace W with a new weight matrix $\tilde{W}$ also based on the degree of similarity between the station i and the corresponding neighboring stations. This new matrix will take into account the distance and also the difference of capacities between a station and its neighbors. The set N_i of neighbors of station i is defined as previously by the stations with a maximal distance d from station i.

In order to measure the similarity degree between two spatial objects, the Euclidean distance is most often used. However, in our case, the use of this distance is inappropriate since the location and capacity attributes are measured on different scales. Hence, we propose to use the Gower's coefficient [Gower, 1971] to calculate the similarity between two stations. Gower's coefficient is a similarity measure which computes the distance between two instances on each attribute k, and then aggregates all of them to finally calculate the similarity degree.

Gower's similarity degree $GOWER_{ij}$ between two stations i and j is defined by:

$$GOWER_{ij} = \frac{\sum_{k=1}^{n} W_{ijk} \times S_{ijk}}{\sum_{k=1}^{n} W_{ijk}} \tag{6.2}$$

where

- W_{ijk} is the weight associated to the attribute k,
- S_{ijk} is the similarity between two stations i and j for the k^{th} attribute, given by

$$S_{ijk} = 1 - \frac{|x_{ik} - x_{jk}|}{r_k}$$

where x_{ik} is the observable attribute k in station i and r_k is a standardization for the attribute k since each attribute is of different unit.

In the context of Velib stations, we calculate the similarity S_{ij} of the location SD_{ij} and capacity SC_{ij} between two stations i and j by:

$$SD_{ij} = 1 - \frac{d_{ij}}{d}$$

$$SC_{ij} = 1 - \frac{|Capacity_i - Capacity_j|}{Capacity_{max} - Capacity_{min}}$$

where

- d_{ij} is the distance between the two stations and d is the maximal distance.
- $Capacity_{max}$ and $Capacity_{min}$ are respectively the maximal and minimal stations capacities in the neighborhood of station i.

In this definition, $W_{ijk} = W_{ij}$ previously defined by equation (7.2.2.1).

We propose in the following to modify the construction of the contiguity matrix of weights by incorporating the spatial and non-spatial attributes and in a weighted manner in the calculation of the weights associated with neighbors. For each neighboring station j, its new weight $GOWER_{ij}$ regarding the station i is given by equation (6.2).

The normalization of the contiguity matrix of weights is done per line, so for each station i, the weight of each neighboring station j is divided by the sum of the weights of all the neighboring stations of i.

Thus, the new contiguity matrix $\tilde{W}$ is given by:

$$\tilde{w}_{ij} = \begin{cases} GOWER_{ij}, & \text{if } 0 \leq d_{ij} \leq d \\ 0, & \text{otherwise.} \end{cases}$$

We applied the improved version of Moran scatterplot to detect the isolated problematic stations. Recall that these stations are defined as spatial outliers with a critical occupancy rate. We used the same dataset described in Section 6.3.

Moran scatterplot representation for the occupancy data of the stations at a fixed timestamp: $10:00\ am$ is given in Figure 6.3. At this time of day, we can expect that the system is highly unbalanced, as in general in a working day a lot of trips take place in the morning around $8:00\ am$. The spatial outliers stations (almost 300 stations) are located in the upper-left and lower-right quadrants. One can notice that there are fewer points in these quadrants compared to the locations with positive correlation.

The number of detected isolated problematic stations at $10:00\ am$, depending on the allowed distance, is given in Table 6.1. Recall that isolated problematic stations are defined as spatial outliers with critical occupancy rate. According to this table, there are about 50 isolated problematic stations at $10:00\ am$. The allowed distance

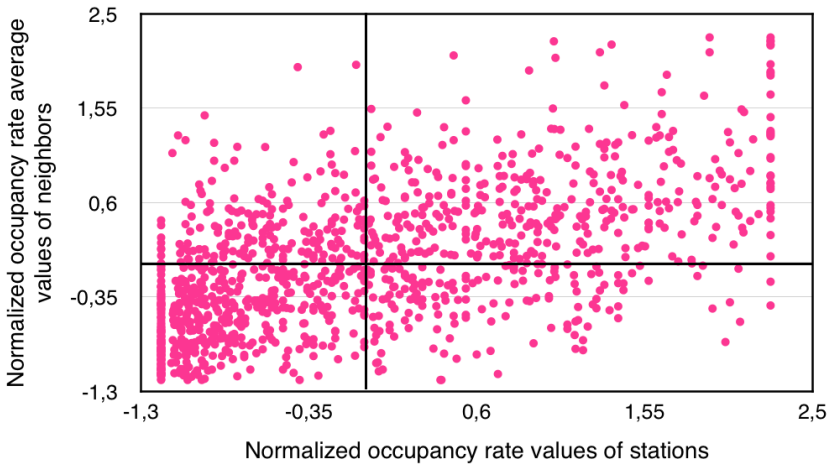


FIGURE 6.3 – Improved Moran scatterplot based on occupancy data of Velib system on Thursday $10/31/2013$ $10:00$ *am*.

does not have a considerable impact on the number of outliers and the isolated problematic stations. Moreover, with a local change of their trips, Velib users can enhance the occupancy rate of about 300 stations (spatial outliers), which represents 24.48% of Velib stations.

TABLE 6.1 – Number of detected outliers stations with the improved Moran scatterplot.

Allowed distance	Outliers	Outliers with critical occupancy rate
300	297	53
400	334	52
500	339	54

6.5 Enhancing resources distribution in Velib system

Our objective is to improve resources' availability in the Velib system by reducing the number of problematic stations. For this purpose, we propose and test in this section a new incentive method, based on a natural and ecological regulation performed by Velib users. The main idea behind the proposed method is to balance the global system by performing small changes in the trips in small local areas. A preliminary study is provided in the previous section to check the existence of several isolated problematic stations. In other words, the aim of this part is show that around a given problematic station (in a distance smaller than 500 meters), there are many balanced stations (with an occupancy rate around 50%), which make it possible for Velib users to balance this problematic station by slightly changing their trips (with an award, extra-time for example).

Using the dataset described in Section 6.3, we plot in Figure 6.4 the evolution of the number of current trips during the day (on Thursday $10/31/2013$), in order to understand the usage of the Velib system. One can easily identify two peaks at about $8:00$ am and $6:00$ pm. They clearly correspond to the trips to the offices and

the return home after work, as it is a working day.

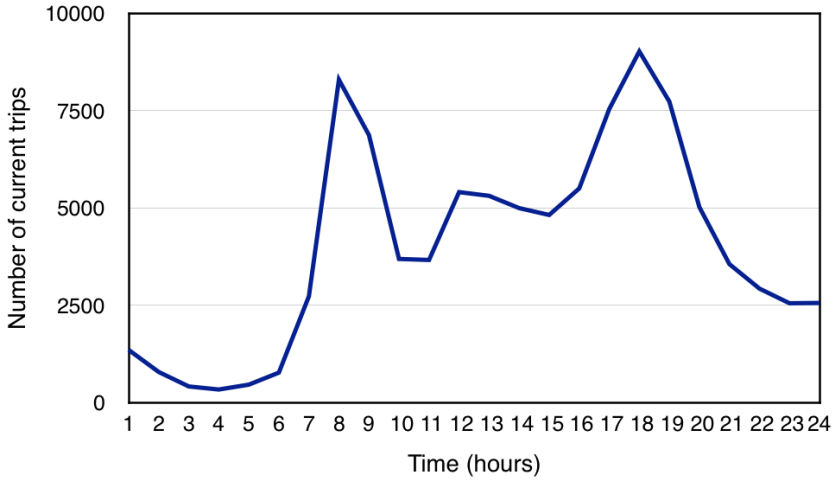


FIGURE 6.4 – Number of trips over time.

Users trips unbalance the Velib system by making some stations problematic (almost empty or almost full). Based on the thresholds of station occupancy introduced before (10% and 90%), the current number of problematic stations is given in Figure 6.5. Despite the performed bike regulation using tracks, the number of problematic stations during the day remains high. The problematic stations are mainly composed of almost empty stations.

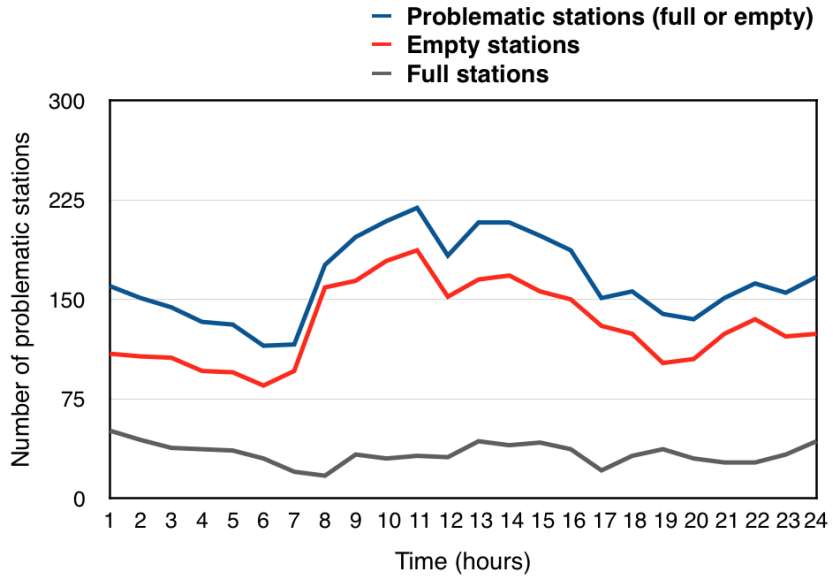


FIGURE 6.5 – Number of problematic stations.

We propose in this section an incentive method that encourages Velib users to improve the homogeneity of the stations in terms of occupancy rate by slightly changing their trips. In the trips dataset, let us denote by A the station where the trip begins and by B where the trip ends. The neighborhood of the station is defined by a distance less than 400 meters. The key idea is to change the trips as follows:

For each trip, in terms of occupancy rate,

- station A will be replaced by the busiest station in the neighborhood of A,
- station B will be replaced by the emptiest station in the neighborhood of B.

The proposed method is inspired by Velib + which consists of offering users of Velib an extra time (that can be cumulated) when they park their bike in a station having a high altitude. The main difference is that Velib+ regulation is static: Velib+ stations are well known and never change over time, whereas our preferred busiest and emptiest stations dynamically change. They vary during the time depending on their occupancy rate and the occupancy rate of their neighboring stations.

Figure 6.6 presents the impact of the proposed incentive method on the number of problematic stations. The results show a clear decrease in the number of problematic stations throughout the day. The average number of problematic stations drops from 164 in real trips to only 27 by slightly modifying each trip. Starting from a relatively high number of problematic stations (almost 150), users are able to balance almost all these stations within three hours. No new trips are either added or lost. The modification is done with exactly the same number of trips. The real trips are only locally modified. The obtained results confirm our intuition that resources global availability in the Velib system can be significantly improved by acting locally. This improvement would allow accepting new trips, where originally users are rejected due to a lack of bikes.

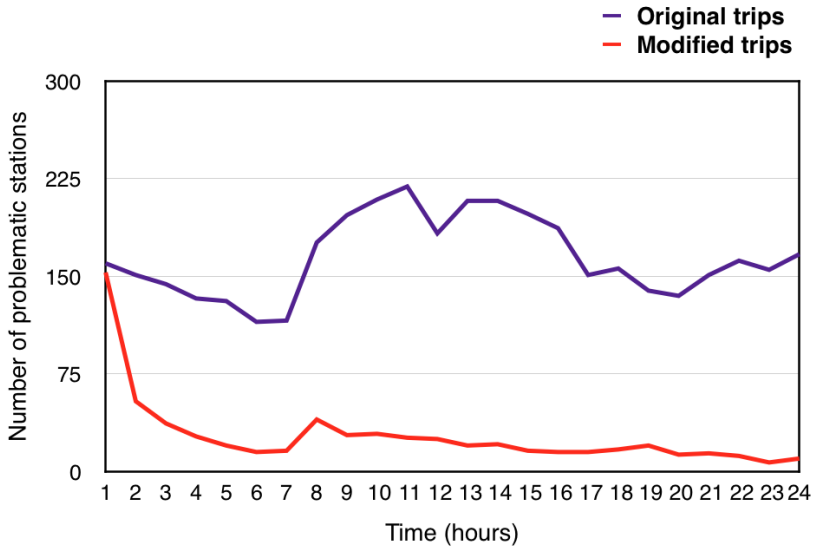


FIGURE 6.6 – Number of problematic stations in the day.

The performance of the proposed incentive method can also be measured by the number of spatial outliers in the Velib system. They consist of stations with an occupancy rate significantly different from the average occupancy rate in their neighborhood. These outliers stations are depicted in Section 6.4 using Moran scatterplot. The comparison of the number of spatial outliers stations between the original and modified behaviors is given in Figure 6.7. With the improved user

behavior, the number of spatial outliers drops significantly, which enhances stations homogeneity in the Velib system.

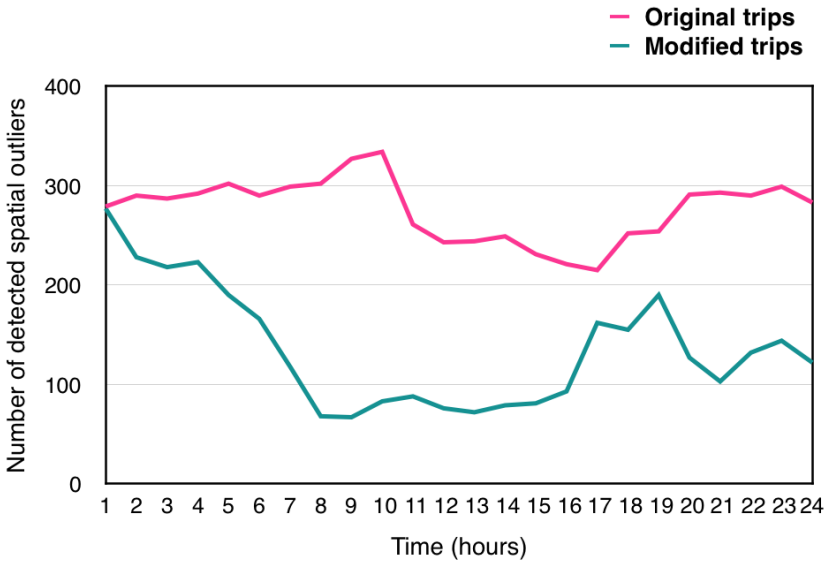


FIGURE 6.7 – Detected spatial outliers.

In Figures 6.6 and 6.7, all users trips are modified according to the proposed method. It is not a realistic scenario as in real life, many users will not accept to change their departure or arrival station even if they are encouraged by a financial motivation or an extra offered time. To simulate a real-world situation, we plotted in Figure 6.8 the average number of problematic stations in the day under a variable collaboration rate of the users. One can see that, if only 20% of users accept to change their trips, the number of problematic stations will decrease by half. The decrease in the number of problematic stations is fast (faster than a linear decrease) which is an excellent result as we cannot expect that the majority of users will collaborate.

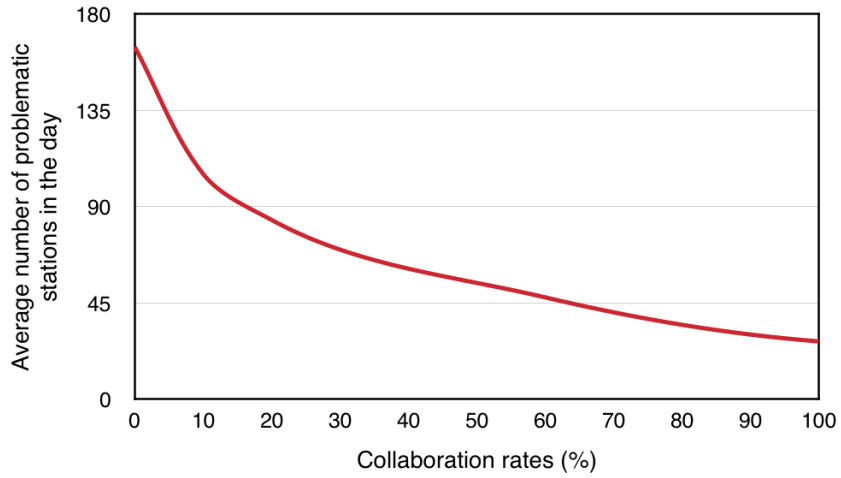


FIGURE 6.8 – Average number of problematic stations in the day after the users collaboartion.

The number of problematic stations during the day is a good indicator to evaluate the quality of the service offered to Velib users. However, it cannot entirely qualify

service availability. For a given station, the service is considered as interrupted if there is no bike or no dock in this station. In this case, the station is said invalid or out of service. Note that this concerns just one resource: bikes or free docks. To have a complete information, we plotted in Figure 6.9 the average duration of stations invalidity during each one-hour interval of the day, before and after the proposed improvement. One can notice that the mean duration of station invalidity has largely decreased, and likewise, the mean cumulative invalidity duration during the day has been widely improved (cf. Figure 6.10). According to Figure 6.10, at the end of the day, the mean invalidity duration of a Velib station drops from 141 minutes to only 22 minutes using our proposed improvement.

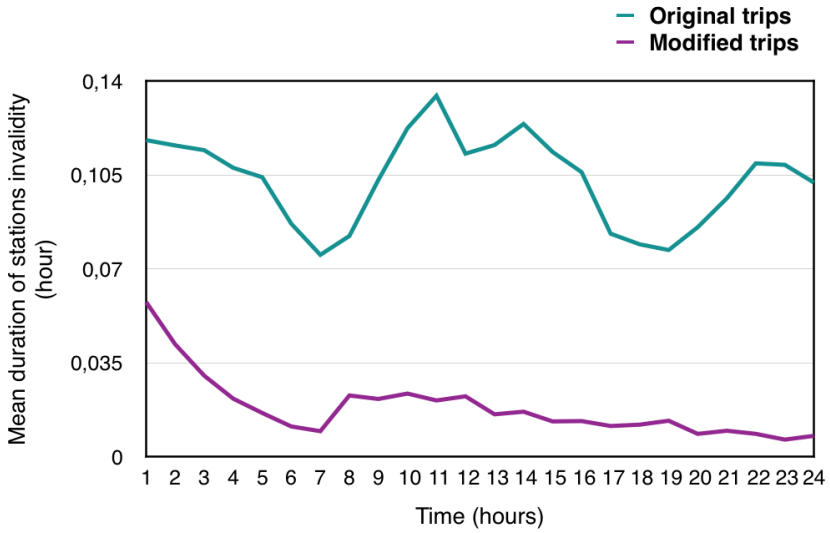


FIGURE 6.9 – Mean duration of stations invalidity.

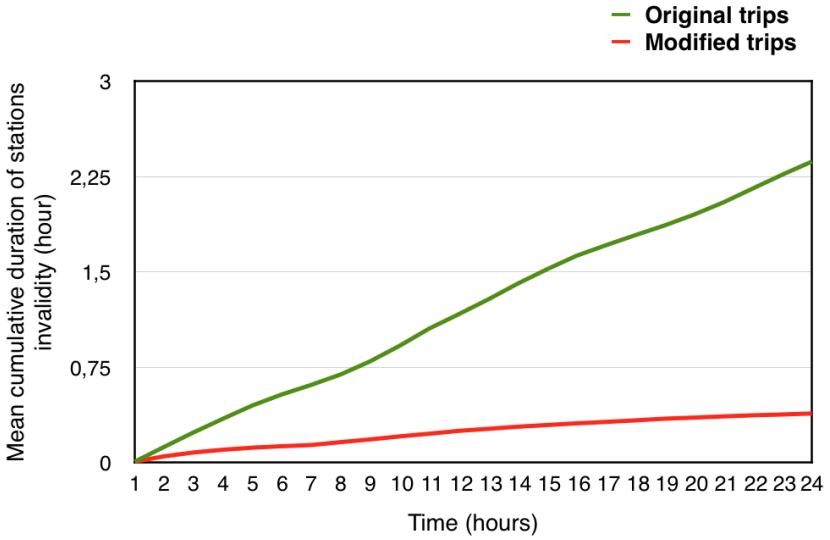


FIGURE 6.10 – Mean cumulative duration of stations invalidity.

6.6 Conclusion

We studied in this chapter the anomalies detection in a spatial context. Our use case was the Parisian bike sharing system (Velib). A Velib station is considered as a spatial outlier if it is almost empty or almost full while the surrounding stations are globally balanced. This is due to the problem of heterogeneity in Velib system causing resources unavailability and users dissatisfaction. To identify spatial outliers in this context, we used Moran scatterplot. In order to calculate the occupancy distance between two stations, we introduced a similarity weight that takes into account the geographical distance between the stations as well as the difference between their capacities. This degree of similarity is calculated using a robust distance metric called Gower's coefficient. Thereafter, we proposed and tested a new algorithm that locally improves the resources distribution (bikes and docks) in the stations, and we experimentally validated its efficiency. The results showed that the proposed algorithm improves the homogeneity of the Velib system by reducing the number of outlier stations and the duration of unavailability of the stations during the day, which ultimately leads to the improvement of the users' satisfaction degree.

In our future work, we aim to detect spatial anomalies in the context of WAVES. We will use Moran scatterplot to study the spatial correlation between several sectors in order to detect the anomalies. In fact, some sectors belonging to the same area (housing area or industrial area) are supposed to present a strong similarity, which can be explained by the periodicity of human activities. First, clustering methods will be applied in order to discover the profile of each cluster. Then, with Moran scatterplot, the anomalies will be identified as sectors with a notable deviation from the standard behavior of their cluster.

Part IV

Data streams native filtering

Chapter

7 Implementation of the data streams native filters solution

Contents

7.1 Introduction **123**

7.2 Implementing data streams sampling and qualification modules in WAVES platform **124**

7.2.1 WAVES FUI project 124

7.2.2 Native filters module 126

7.3 Information technology infrastructure for data streams native filtering **129**

7.3.1 Resources consumption in native filters 129

7.3.2 Moving to the Cloud computing 137

7.4 Conclusion **142**

The scientific contribution and the obtained results presented in the third section of this chapter have been published in our paper: **Rayane El Sibai**, Yousra Chabchoub, Raja Chiky, Jacques Demerjian and Kablan Barbar. *"Information Technology Infrastructure for Data Streams Native Filtering"*. In the IEEE Middle East & North Africa COMMunications Conference, IEEE, 2018.

7.1 Introduction

We recall that our work is a part of the FUI 17 WAVES project. This project is motivated by the water leakage problem in the water distribution network. To supervise the network, several flowmeters are deployed in a large geographical area. They send many observables related to the water in this area to the monitoring system. This latter proceeds then to detect and to analyze the abnormal data. Its

objective is to filter useful information and to infer new knowledge in order to help the environment explorer to make right decisions to repair the leaks if they exist. As we discussed in the previous chapters, this whole process, from the data collection to the data analysis leads to two key problems: data volume and data quality. One solution to overcome these problems is to summarize and to clean the received data.

In this thesis, we are responsible for the native filters module of the WAVES platform. This module provides two features: real-time data qualification and data summarization. Thus, the native filters module is composed of two sub-modules: qualitative filter and quantitative filter. Upon receipt of the data streams from several sensors, the qualitative filter sub-module evaluates and improves the quality of the data based on the architecture presented in Chapter 4. In particular, it improves both quality dimensions: accuracy and completeness. It detects, deletes and replaces the outliers, estimates missing and deleted data and deletes duplicated data. It also adds several quality statistics to the raw data. Once qualified, the data are summarized by the quantitative filter sub-module. This later implements the following sampling methods: Chain-sample, Deterministic sampling, Simple Random Sampling and Reservoir sampling. The native filters module processes several streams at the same time and adapts to the characteristics of each stream. Several configurations are possible to allow the user a flexible manipulation of the module to meet his needs.

This chapter is organized as follows. In Section 7.2, we present the context and the global architecture of the WAVES platform, as well as the native filters module. In Section 7.3, we study at first the computational resources of the native filters solution, secondly, we discuss its migration to the Cloud computing environment. We end the chapter with a conclusion.

7.2 Implementing data streams sampling and qualification modules in WAVES platform

7.2.1 WAVES FUI project

This thesis is part of the FUI 17 WAVES project, which aims to design and to develop a monitoring platform for the supervision of water distribution networks. The goal is to develop a relevant solution for the realization of a decision support system for the network operators and explorers. For example, one may be interested in detecting abnormal phenomena (micro-variations of certain parameters that have an impact in terms of risk, variation or non-nominal frequency, etc.) as soon as possible, thus, allowing to save considerable amounts of potable water. The remote-monitoring of the water consumption, the numerous communicating sensors recently deployed in the water network, as well as the data coming from social networks, generate new data streams that will make it possible to diagnose leaks, breakdowns or accidents quickly and accurately.

The input data of the decision-making system are heterogeneous (different raw formats such as CSV, XML, RSS, or JSON) since they come from different sources (sensors, social networks, customer complaints, static descriptions of the network

and sensors, etc.). In order to be able to process and to reason optimally these data, semantic web tools such as Resource Description Framework (RDF), Web Ontology Language (OWL), or SPARQL can be used. The objective is to design and develop a platform that "semantises" static and dynamic data from several heterogeneous sources, filters them, summarizes them and proposes reasoning tools. This platform will provide a decision support solution for operators to pilot networks in real-time and enrich their knowledge base through reasoning (inference). For example, based on the system decision, the network operators may take action in a site before a phenomenon gets worse, start maintenance of a drifting sensor, or diagnose a sensor that no longer communicates.

Since the considered data streams are permanent and arriving at a high speed, they produce a huge mass of data, impossible to store entirely in due time. Erroneous, inaccurate or inconsistent data cause considerable damage in the case of supervision and detection of abnormal phenomena in water transport and distribution networks. It is, therefore, necessary to filter these data on the fly and to store only those that are relevant by producing summaries. This is the purpose of data streams native filtering.

Once filtered, it is essential to reason in a context of streaming. Semantic data streams must be interconnected and correlated to extract relevant information that can, in turn, take the form of new data streams. This mechanism of inference and exploitation of the filtered semantic data must support the scalability in terms of the stream rate and the volume of the data to be processed. It is very important to take into account the heterogeneity of flows in terms of arrival rate and regularity.

Reasoning on a semantic stream represents a challenge for the conception a semantic data stream processing platform. It is important to ensure the accuracy, integrity, authenticity, reliability, and consistency of this extracted information even after being filtered, summarized, or interconnected with other streams.

The global architecture of the WAVES platform is presented in Figure 7.1.

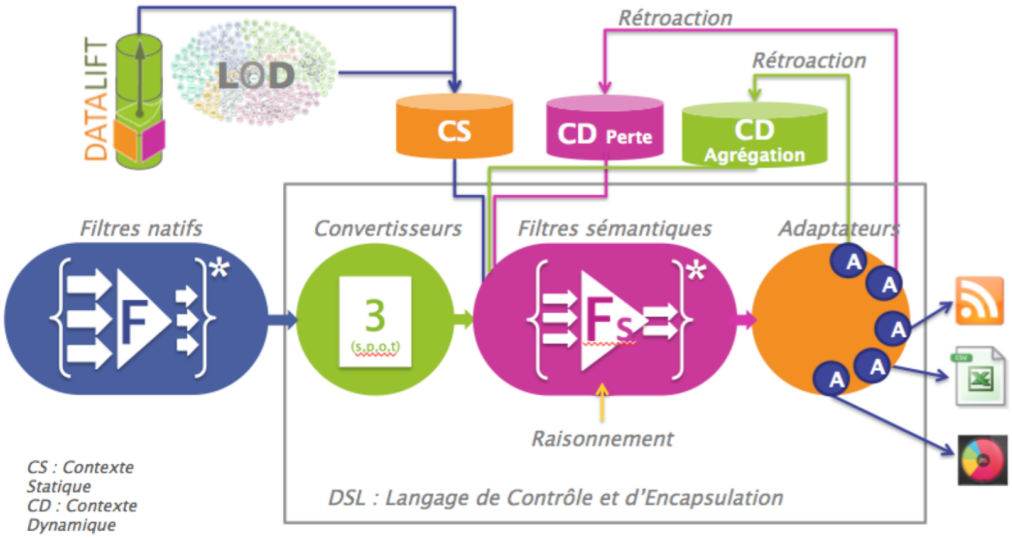


FIGURE 7.1 – WAVES platform architecture (from FUI17 WAVES Annexe technique).

7.2.2 Native filters module

The native filters module takes as input several data streams, issued from flowmeters, in their native format (CSV, XML, etc.) and gives in output new data streams of the same format. Each sensor reading consists of two fields: a timestamp t designating the registration date of the reading, and a value corresponding to the water consumption volume at time t. Those readings are regularly sent in real-time as soon as they are generated,

This native filters module consists of two sub-modules: qualitative filter and quantitative filter. The qualitative filter sub-module cleans the data by removing the errors and duplicated data and replacing missing and erroneous data. The quantitative filter sub-module is responsible for reducing the large volume of data by using several sampling algorithms. The architecture of the native filters module is given by Figure 7.2.

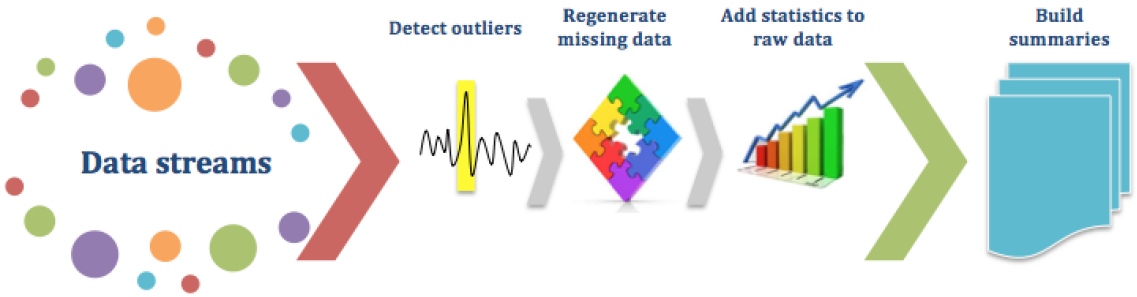


FIGURE 7.2 – Native filters architecture.

7.2.2.1 Qualitative filter sub-module

In the qualitative filter sub-module, the data quality measures are considered as metadata and are propagated with streaming data at the output of the sub-module. The propagation of the data quality measures consists of enriching each data with a data quality vector containing the data quality measures. Thus, the information related to the data quality dimensions will be added to the data stream at the output of the qualitative filter module. Notice that the quality of the data is estimated and enhanced continuously with the arrival of streaming data.

Different flowmeters can have different rates. Thus, the quality measures have not to be recorded and enhanced at the same time for all the received streams. It is, therefore, necessary that the qualitative filter sub-module adapts to the characteristics of each stream, especially, its frequency denoted by $streamRate$. For each flowmeter, the $streamRate$ value is calculated automatically based on the first data arriving from this flowmeter. The size of these data is defined by the $timeIntervalTraining$ parameter.

The data quality dimensions to be evaluated and enhanced are the completeness, accuracy, sensor reliability, and confidence dimensions. Notice that the evaluation and improvement of the precision dimension are out of the scope of our work in this thesis.

Completeness dimension: Data completeness is affected by network congestion and sensor failure. It depends mainly on the frequency for recording data in the environment and the frequency for communicating the recorded data to the data center or the server. Therefore, based on the known stream arrival rate, the amount of missing data at each instant t (in $hour$) can be computed as follows:

$$cumulativeMissing_t = [\frac{\sum_{t_0}^{t} Number\ of\ missing\ data}{(t - t_0) \times streamRate}] \times 100\%$$

In order to avoid taking into account very old values of the stream, $cumulativeMissing$ can be initialized every day as an example. In this case, t_0 refers to the beginning of the current day ($t = 0h$). The initialization period is an input of the algorithm and can be configured by the user. The $streamRate$ parameter is related to the data frequency of the considered flowmeter (in $items/hour$).

Accuracy dimension: The accuracy dimension measures the correctness of the data. As discussed in Chapter 4, this dimension is affected by the erroneous data recorded by the sensor. Sensors errors can be of several types (outliers, spikes, and stuck-at). We are interested in this work into the detection of outliers and spikes. These errors are detected using a static rule that can be defined according to the application domain. In the context of water distribution network we define the static rule as follows:

Any reading value that exceeds the maximum theoretical flow value (called $debitMax$) that a flowmeter can emit will be considered as erroneous of type outlier. The $debitMax$ value is calculated using the Hazen-Williams formula. Indeed, the Hazen William equation determines the maximum transportable flow rate through a given pipe according to the physical properties of the pipe and the pressure drop caused by the friction of the used material.

The $debitMax$ value is calculated according to the Hazen-Williams formula as follows:

$$debitMax = 0.28 \times ((DH/L)^{0.54}) \times (C \times D)^{2.63} \quad m^3/sec$$

where:

- DH is the altitude difference between the two extremities of the pipe, in $meters$.
- L is the length of the pipe, in $meters$.
- C is the friction coefficient of the used material.
- D is the diameter of the pipe, in $meters$.

For each sensor reading, the accuracy degree also called $currentError$, takes the following values:

$$currentError = \begin{cases} 0, & if\ the\ sensor\ reading\ >\ debitMax. \\ 1, & otherwise. \end{cases}$$

Notice that, for each flowmeter, the value of the maximal flow (called $debitMax$) is stocked as a static value in the configuration file of the qualitative filter sub-module.

Sensor reliability dimension: The sensor reliability dimension is measured by the $cumulativeError$ degree which depicts the percentage of cumulated erroneous values. Just like the $cumulativeMissing$, the $cumulativeError$ degree is initialized after a certain duration called $initializeParameter$. This parameter is configurable in the configuration file of the sub-module. At each instant t, the $cumulativeError$ degree is calculated as follows:

$$cumulativeError_t = [\frac{\sum_{t_0}^{t} Number\ of\ erroneous\ data}{\sum_{t_0}^{t} Number\ of\ received\ data}] \times 100\%$$

Confidence dimension: The $confidence$ degree of each sensor reading is set to 100% if the value is original (not regenerated), otherwise, it is set to the proportion of consecutive missing or erroneous values around the considered value. The $confidence$ degree is calculated as follows:

$$confidence = [1 - \frac{m}{interpolationAllowedDuration \times streamRate}] \times 100\%$$

where m is the number of consecutive missing values, $interpolationAllowedDuration$ (in $hours$) is the maximum missing data period allowed to replace the data.

We can notice that at the output of the qualitative filter sub-module, each sensor reading can have one of these three statuses:

- The reading is original if the $confidence$ degree is equal to 100%.
- The reading was missing and replaced if the $confidence$ degree is $< 100\%$ and currentError equals 0.
- The reading was erroneous and replaced if the $currentError$ value is equal to 1.

Once the data quality dimensions are evaluated, the qualitative filter sub-module proceeds to enhance the data quality by deleting erroneous data, removing duplicated data and replacing missing data. In fact, deleted erroneous data are considered as missing data. The regeneration of missing data depends on their duration. When the duration of successive missing data is less than or equal to the $interpolationAllowedDuration$ value, the linear interpolation will be used to impute missing data. The interpolation can not be performed for an important number of consecutive missing or erroneous data. Therefore, if the duration of successive missing data exceeds the $interpolationAllowedDuration$ value, missing data will not be regenerated.

7.2.2.2 Quantitative filter sub-module

When the data is qualified, the quantitative filter sub-module proceeds to summarize them. Sampled data can be either stored in the native format of the input data (csv, XML, etc.), or sent to the next modules of the WAVES platform to undergo further processing such as semantization. Four sampling methods can be used to summarize the data: Deterministic sampling, Simple Random Sampling, Chain-sample and Reservoir sampling. Different sampling parameters can be applied. The

parameter associated with each sampling method is defined in the configuration file of the sub-module as follows:

- For the Deterministic sampling, the sampling parameter is called $jump$, it is an integer $\in [1, x]$, where x is a positive integer.
- For the Simple Random Sampling and Chain-sample, the sampling parameter is called $percentage$, it is an integer $\in [1, 100]$.
- For the Reservoir sampling, the sampling parameter is called $reservoirSize$, it is an integer $\in [1, Size_{max}]$ such that $reservoirSize$ is the size of the summary.

The choice of the sampling parameter is critical, it depends on the use case (application domain), the requirements of the application, the nature of the data, and the level of precision required by the application.

7.3 Information technology infrastructure for data streams native filtering

In order to deal with the huge volume and bad quality of data streams generated by environmental sensors, we presented in this thesis a solution called native filters. It consists of cleaning and summarizing the received data streams. The implementation of this solution was presented in Section 7.2. In the following, we evaluate the efficiency of this solution as a function of the computational resources that it requires. We present a benchmark of the computational resources requirements while examining different infrastructures. Our consideration of the computational resources covers the running time to process (cleaning and summarization) the input data and the amount of memory needed to store the output data (data streams summaries).

7.3.1 Resources consumption in native filters

7.3.1.1 Dataset description

We use the dataset presented in section WAVES dataset. These data are issued from flowmeters delivering water to a big French city. We recall that the duration of each stream if of 21 months, with approximately 60000 recorded observations. These data are generated regularly by the sensors with a frequency of one observation each 15 minutes. Each recorded observation contains two fields: the timestamp designating the recording date of the measure, and the attribute value representing the delivered water volume in m^3.

For the sampling process, without loss of generality, we choose a sampling rate equal to 50%. Thus, the sampling parameters are:

- Number of elements to sample $k = 5$ and window size $n = 10$ for both SRS and Chain-sample.
- $n/k = 2$ for Deterministic sampling.
- $k = 30000$ for Reservoir sampling.

7.3.1.2 Execution time

We study in this section the execution time of the native filtering processes: data cleaning and summary construction, according to the number of received data streams.

Two server units are considered. The local machine server and Rackspace Cloud server. Rackspace Cloud is a set of Cloud services provided to the users with costs calculated based on the utility of the American company Rackspace. The services include Cloud Platforms, Cloud Storage, and virtual private servers. We use Rackspace server as a service. The chosen Cloud server specifications are: RAM: 2 GB, vCPUs: 2, System Disk: 40 GB SSD and Bandwidth: 400 Mb/s. The specifications of our machine are: RAM: 4 GB, System Disk: 120 GB and Processor: 2.7 GHz Intel Core i5.

The execution time of the sampling process includes the time of reading, sampling and writing the data in the summary, and is dependent on the following factors:

- Number of received observations for each stream, which depends on the stream rate or frequency of the sensor.
- Number of streams received simultaneously.
- The sampling rate.
- The size of the window.
- The sampling algorithm.

We display in Figures 7.3 and 7.4 the execution time taken by each one of the sampling algorithms previously discussed to construct the summaries for the received data streams. We notice that the execution time increases linearly with the increase of the number of streams to be processed, and it can be described using the following linear equation:

$$Execution\ time = a \times Number\ of\ streams + b$$

where b is a fixed duration related to the initialization of the quantitative filter responsible for the sampling process.

Figures 7.3 and 7.4 show that Chain-sample has the highest execution time compared to the other sampling algorithms. This is due to the collisions problem. We recall that the window size has a high impact on the execution time taken by Chain-sample. This time increases as the window size increases, as we have seen in Chapter 2.

Figure 7.5 depicts the execution time of the cleaning process using the local server and the Cloud server. According to this figure, the cleaning time is not linearly dependent on the number of streams to be processed. Actually, for a single stream, this time depends on several parameters: the number of received observations, the amount and the distribution of missing and outliers data. As these parameters vary widely from a stream to another, the cleaning time cannot be predicted in advance.

Increasing the availability of a system involves maximizing the percentage of time during which the system is operational. In order to protect the system against unexpected overloads and in order to avoid the system failures resulting from the number of data streams received, load balancing can be used to ensure high avail-

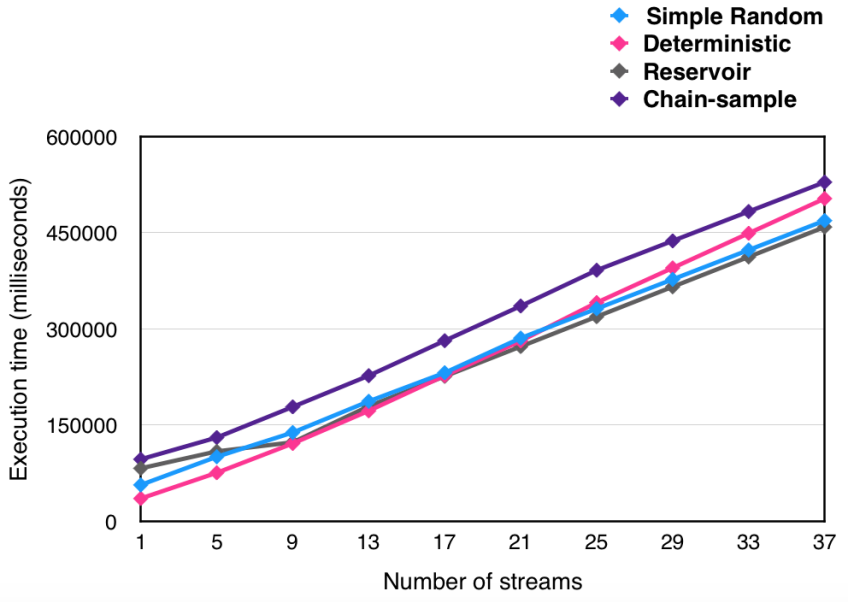


FIGURE 7.3 – Execution time of the sampling process according to the number of streams, using the local server.

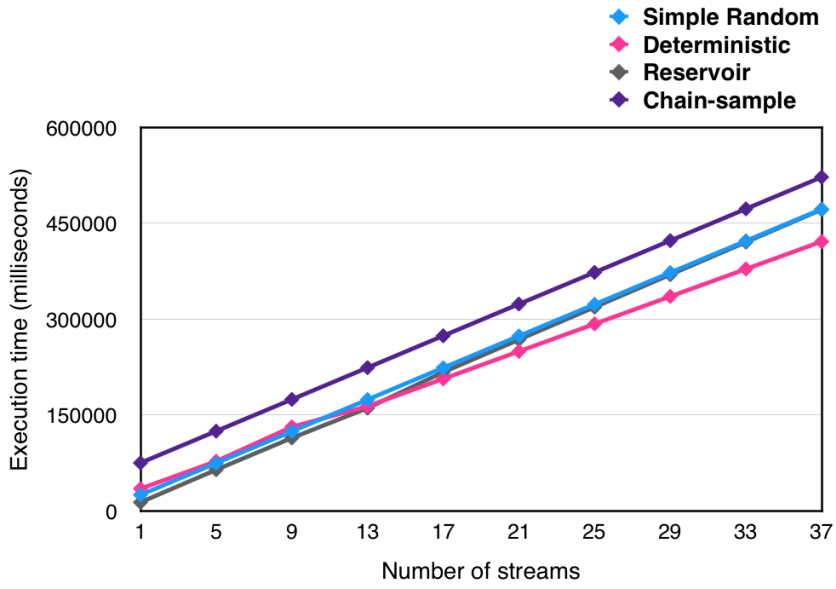


FIGURE 7.4 – Execution time of the sampling process according to the number of streams, using the Cloud server.

ability. It consists of using a set of servers in which the incoming streams are evenly distributed to help reduce the load on a single server.

In order to predict the needed number of servers to ensure high availability, we calculate the number of observations that a server can process in 15 minutes (corresponding to the sampling frequency of the sensor), knowing that each received observation corresponds to a single stream. We denote by this number as the computing capacity per server. Regarding the sampling process, this number is presented in Table 7.1 depending on the used sampling algorithm. According to this

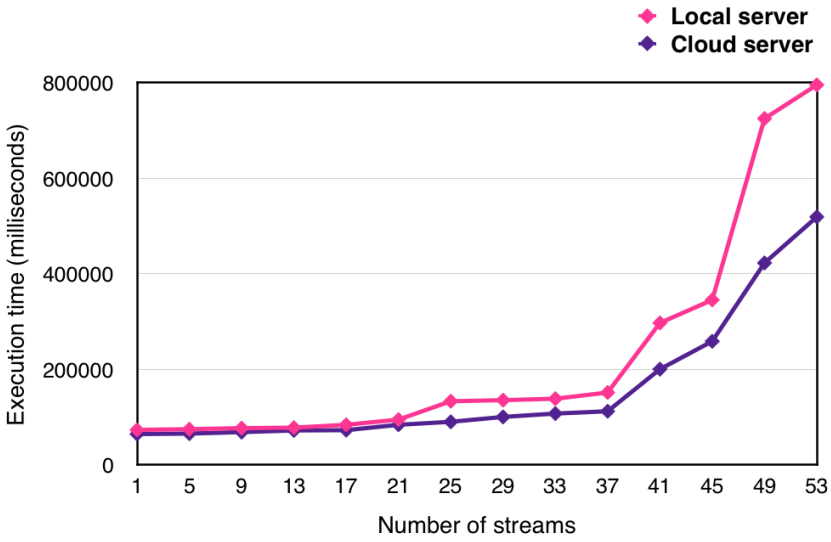


FIGURE 7.5 – Execution time of the cleaning process according to the number of streams.

table, the number of servers needed to ensure high availability can be calculated using Equation 7.1. The obtained values using the Cloud server are shown in Figure 7.6. Notice that the number of servers is always rounded.

TABLE 7.1 – Number of observations that can be treated in 15 minutes, using the Cloud server.

	SRS	**Deterministic**	**Reservoir**	**Chain**
Local server	1M	900K	1.3M	700K
Cloud server	2M	1.5M	3M	1M

$$Number\ of\ servers = \frac{Number\ of\ received\ streams}{Computing\ capacity\ per\ server} \tag{7.1}$$

Notice that the sampling rate varies widely by application. For instance, a WSN of 90 measurement points were implemented to detect forest fires [lib, 2010]. Each measurement point is connected to 4 sensors with a sampling frequency of 1 sample per 5 minutes. Using the observations shown above, we can deduce that this network requires 0.1% of the local server's capacity for native filtering, thus, very low utilization rate and profit-to-cost ratio. High availability worsens this situation since even less utilization rate will be achieved. Native filtering for low-rate sensor networks is very costly in traditional computing environments.

In [Küçük et al., 2015], the sampling frequency of the sensor for electrical energy monitoring is 3200 samples per second. The number of sensors/streams is not known, but we can deduce that for every sensor 1 local server is needed. In such a case, a large data center is required to handle the native filtering processes. Additional costs including certified personnel, site licenses and disaster recovery are added to the overall cost making it a substantial investment. Native filtering for high-rate sensor networks is very costly in traditional computing environments.

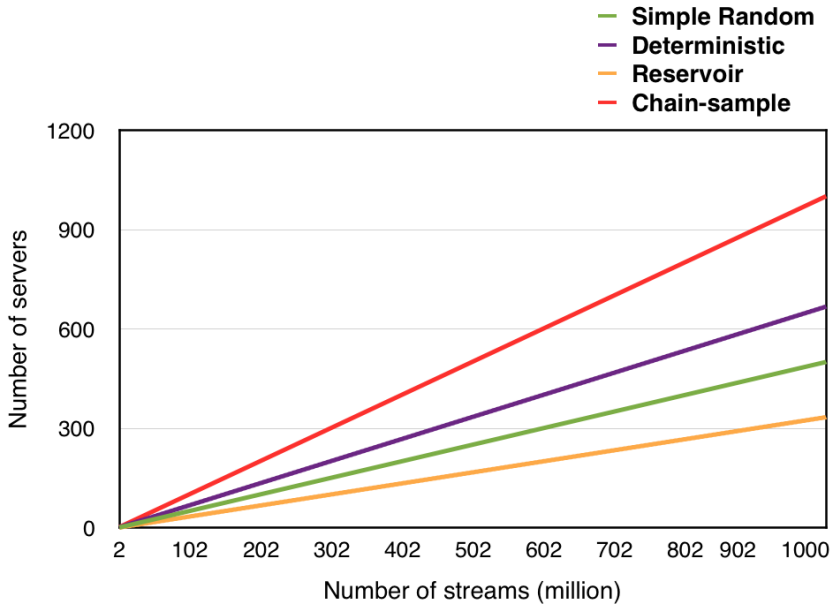


FIGURE 7.6 – Number of servers needed to ensure high availability, using the Cloud server.

7.3.1.3 Data storage models

Querying the summary requires its permanent storage on the disk. Without ensuring high availability, several choices can be made for the preservation of the summary, among them, the Database Management System (DBMS) and Hadoop Distributed File System (HDFS). The size of the summary (number of elements) to be stored can be represented as a function of the number of elements k to sample on each window, the size of the window n, the size of the received data DS, the amount of excess/lack of sampled elements δ, the step n/k and the number of received streams x. Assuming that the sampling rate k/n is the same for all the received streams, the summary size is given in Table 7.2 according to the used sampling algorithm.

According to Table 7.2, the size of the summary provided by SRS algorithm is not exactly proportional to the size of the stream. There is always some excess or lacked elements in the summary. This is due to the randomness in SRS. The size of the sample over a window of size n is a random variable that can be theoretically modeled by the binomial distribution, as explained in Chapter 2. The size of the sample over a window is variable $\in [0, n]$ with an average of k (with a sampling rate of k/n).

With the Chain-sample algorithm, since the windows are sliding, a sampled element belongs to a minimum of 1 sample corresponding to one window, and a maximum of n samples corresponding to n successive windows. Hence, the size of the summary S_f is always $\leq k \times W$ where W is the number of sliding windows. In addition to the number of sampled elements over each window, k, the size of the summary depends also on another parameter that we call het. This value depicts the heterogeneity between two successive sliding windows. It represents the number of new sampled elements in the current window, compared to the window just before. It can be also considered as the difference of the samples related to two successive sliding windows. It is a variable $\in [0, 1]$. The higher the heterogeneity value, the

higher the size of the summary.

TABLE 7.2 – Storage requirements of data streams summaries according to the sampling algorithm.

Sampling algorithm	Summary size	Summary size (number of rows)
Chain-sample	Unlimited	$x \times (((DS \text{ - } n) \times het) + k)$
Reservoir	Limited	$x \times k$
SRS	Unlimited	$x \times (\frac{k}{n} \times DS + \delta)$
Deterministic	Unlimited	$x \times \frac{k}{n} \times DS$

Using a Database Management System

The use of a DBMS offers many advantages concerning the manipulation of the summary. Firstly, in a database management system, searching and accessing the data is easier and user-friendly because of the predefined queries. Secondly, the DBMS has a high-security system by using the encryption and the biometric security measures. Last but not least, the DBMS deals with the concurrent access of the data by using the locks.

We show in Figure 7.7 the relational schema considered to model a data stream summary. Because of the growing and infinite size of the stream, the summary size often increases as the data arrive. Very few sampling algorithms such as Reservoir sampling [Vitter, 1985] construct a fixed size summary independent of the size of the stream. In this case, the summary is always updated to replace the old elements.

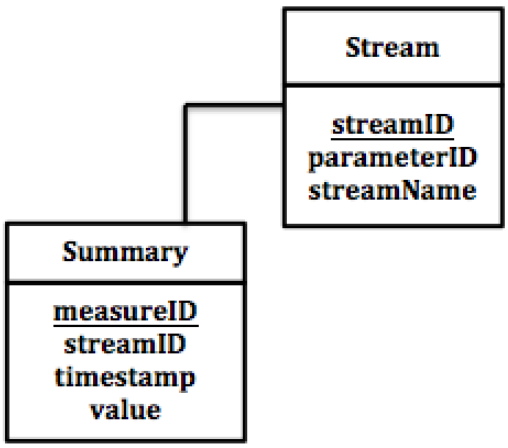


FIGURE 7.7 – Relational model of the summary.

Estimating the size of the database involves the examination of the size of each table. For this purpose, the physical storage requirements for each row must be calculated. The sum of tables' sizes represents the total size of the database. A database is a collection of several pages spread across one or more physical files. A page is the smallest data storage unit in Microsoft SQL Server. It is of size 8 KB and used to store the data rows of the database tables. Notice that 8 KB correspond to 8192 bytes, however, 96 bytes are allocated to the header page. Thus, 8096 bytes can be stored in the page [Patton and Ogle, 2001].

The size of the table $summary$ is dependent on the row's length in the table and the number of observations (each observation corresponds to one row) chosen by the sampling algorithm to be added to the summary. The storage space of the table $summary$ is calculated as follows [Laird et al., 2001]:

$$\begin{aligned} &Storage\ requirements\ (bytes)\ per\ stream \\ &\quad = Number\ of\ pages \times 8096 \\ &\quad = \frac{Total\ number\ of\ rows}{Number\ of\ rows\ per\ page} \times 8096 \end{aligned} \tag{7.2}$$

The number of pages required to store the rows of the table $summary$ is calculated as the total number of rows of the table divided by the number of rows that one page can store. Actually, 8096 bytes can be stored in one page. To calculate the number of rows that one page can store, we need to calculate the length of each row in bytes. Each row is composed of one float value ($value$) and two integer values ($measureID$ and $streamID$) and a timestamp. Given that the size to store an integer value or a float value of 4 digits is 4 bytes and the size to store a timestamp is 8 bytes, the total size of each row will be equal to $4 + 4 + 8 + 4 = 20$ bytes.

Therefore, one page can store approximately 404 rows. Consequently, the number of pages required to store 1440 records per month (number of records obtained using a sampling rate equal to 50% with Deterministic or Reservoir algorithms) for a single stream is 3.56 pages. Therefore, the storage requirement of the table $summary$ is $3.56 \times 8059 = 0.000026$ GB per month. By multiplying this number by 21, we obtain the storage requirements of the summary for 1 year and 9 months, which is equal to 0.00056 GB for a single stream, and to 0.038 GB for the 68 streams corresponding to the 68 flowmeters.

Using Hadoop

Hadoop Distributed File System (HDFS) is another choice that can be taken to store the summary of a data stream. Unlike the traditional database management systems that do not have the capacity to handle large amounts of data, Hadoop is a scalable storage platform that stores and distributes very large amounts of data across multiple servers running in parallel. Several storage techniques can be used in Hadoop: Apache Avro, Apache Parquet, Apache HBase and Apache Kudu.

We study the storage space requirements for a data stream summary in HBase. Our choice is motivated by several reasons. HBase is suitable for streams environment. It allows writing data very fast with a very low latency in a real-time manner. The data can be read randomly, thus reducing the response time to requests, while giving the possibility of modifying the stored data. The representation of the table $summary$ and the structure of each row in this table are shown in Figure 7.8.

In order to calculate the storage size needed, we have to calculate the size of each row. For a single stream, the required storage size of the summary in HBase is given by:

$$storage\ size = R \times S \times r$$

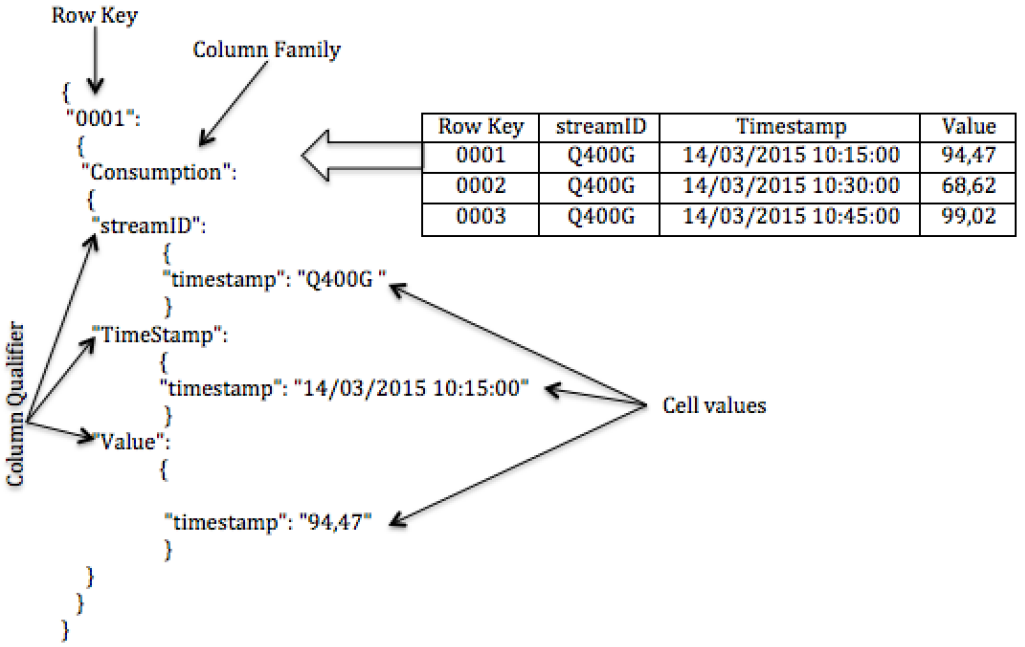


FIGURE 7.8 – Representation of a row in an HBase table as a multidimensional map.

where R is the number of rows, S is the row size and r is the replication factor. The replication factor provides fault tolerance. If a copy of the data is not accessible, then, the corrupted data can be read from another copy. The default replication factor is equal to 3. HBase is a column-oriented database. It stores the data in a key-value format. So, each value is stored with a fully qualified row key. The size of a row is calculated as the sum of the fixed part and the variable part of the row. The size of the fixed part is equal to:

$KeyLength + ValueLength + RowLength + CFLength + Timestamp + KeyValue = 4 + 4 + 2 + 1 + 8 + 1 = 20$ bytes.

The size of the variable part is calculated as the sum of the bytes array sizes of Row, $ColumnFamily$, $ColumnQualifier$ and $Value$ values. Assuming that all the received data have the same structure as the data with $RowKey = 0001$, the size (in $bytes$) of each column value of each row is calculated as follows:

- Second column (streamID): $20 + 4 + 11 + 8 + 5 = 48$ bytes.
- Third column (Timestamp): $20 + 4 + 11 + 9 + 19 = 63$ bytes.
- Fourth column (Value): $20 + 4 + 11 + 5 + 5 = 45$ bytes.

Thus, the size of each row is equal to 156 bytes, and so, the space required to store a data stream summary over a period of 21 months is equal to $0,0043 \times 3 = 0,013$ GB (using a sampling rate equal to 50% and Deterministic or Reservoir sampling), and thus, equal to 0.884 GB if 68 streams are received. Recall that the value 3 is the replication factor. One can notice that the space required to store a summary of x data streams in HBase is 23.35 times more than that in DBMS.

Data compression can speed up the read/write operations and save the storage space in HBase. There are several compression solutions such as Snappy, ZLIB, LZO and LZF with which around 40% of compression ratio is achieved [Shriparv, 2014]. However, this mechanism increases the processing time and the CPU utilization because of the decompression.

7.3.2 Moving to the Cloud computing

In this section, we discuss the migration of data streams native filtering process to the Cloud computing environment.

7.3.2.1 Cloud computing basic concepts

Cloud computing definition

Cloud computing, known as on-demand service is an Internet-based computing where IT resources and Software applications are provided to computers and mobile devices on-demand. Its main concept is: "Why would you buy anything when you can rent it ?". Thus, instead of investing in infrastructure, users and businesses may find it useful to rent the infrastructure and the needed software to run their applications. In this environment, there are service providers that facilitate, manage and render the services to the users and businesses who in their turn will pay the costs of the leased services. Cloud computing provides two basic functions as services: computing and storage. With this technology, the users and businesses can access programs, storage and application development platforms through the Internet and via the services offered by the Cloud computing providers [Armbrust et al., 2010].

The adoption of the Cloud environment has several benefits. It allows users and businesses to save time and costs. In fact, companies that manage their own platforms by themselves must buy and maintain their hardware and software infrastructures. This requires human resources with professional knowledge and special skills to take care of the platforms. With the use of Cloud computing, the cost of storage has dropped dramatically and the efforts of the infrastructure installation, configuration, and maintenance have been overlooked. In addition, the estimation and planning of the required resources, as well as the use of excessive storage and computation capacities solely to manage maximum workloads are no longer required as the resources can be flexibly adjusted as needed.

Types of Cloud services

- Software as a Service (SaaS): It represents the capability provided to the Cloud users to use and to run the applications on the Cloud. These applications are accessible by the users through a web interface.

- Platform as a Service (PaaS): It is a development platform that enables the users of the Cloud to develop services and applications directly on the Cloud. An example of PaaS is Google AppEngine.

- Infrastructure as a Service (IaaS): It consists of providing the users of the Cloud several computing, storage and network resources using which the users can run their own applications or software.

Cloud deployment models

Cloud deployment models can be categorized into four types: public Cloud, private Cloud, hybrid Cloud and community Cloud [Mell et al., 2011, Nepal et al., 2015].

- Public: The public Cloud is the popular Cloud deployment model. With the public Cloud, the service provider is the owner of the Cloud and anyone can access its services through web interfaces. The access to the services is paid and only for the duration during which the services are used. Many popular Clouds adopt the public deployment mode, such as Amazon EC2, S3 and Google App Engine.

- Private: The private Cloud can be compared to the Intranet, it is owned by the company in which only the authorized users can access the services provided. Unlike the public Cloud where the resources and applications are managed by the Cloud provider, the services in the private Cloud are managed by the organization itself. The main benefit of this model is the security aspect, in particular, the confidentiality of the data, that is enhanced as only the users in the organization have access to the Cloud.

- Hybrid: The hybrid Cloud is a composition of several Clouds whose infrastructures are distinct. It is the mix of public and private Clouds. Notice that the fact of using two types of Cloud (public and private) at the same time cannot be considered a hybrid Cloud. In fact, the Clouds must be used in conjunction with each other. For instance, an organization can use a public Cloud that processes the data and sends it to a private Cloud for storage. In such a case, the Cloud is considered hybrid.

- Community: A Cloud of type community is a collaborative Cloud computing solution targeted to a limited subset of individuals or organizations. This shared Cloud is governed and managed commonly by all the participating organizations or by a third-party. This type of Cloud is usually used by organizations working on joint projects or research and requiring a shared platform for managing and executing their projects.

7.3.2.2 Cloud computing criteria: decision factors

Migrating the native filtering processes to the Cloud depend on the value of the additional features provided by Cloud computing in comparison to the expense resulting from this move. The decision factors influencing the adoption of the Cloud computing environment are the following:

- Data security: Cloud computing can save time and costs, but it is more important to trust the system. One of the biggest obstructions in the Cloud

computing is the data security. In fact, the data of the users are dispersed across multiple machines and storage devices such as servers and computers and various mobile devices such as wireless sensor networks and smart phones. This makes the security of the data in the Cloud computing quite serious. In order to ensure the reliability of the Cloud and the trust of the users regarding this environment, it is necessary to protect and ensure the security of the data stored in the Cloud.

The trust in a Cloud environment depends on the deployment model and on the data protection and prevention techniques that are used. In a public Cloud deployment, the control of the data access is delegated to the organization owning the infrastructure who is responsible to define a security policy. Data security is the combination of the integrity, anonymity and confidentiality of the data in the Cloud. Data integrity preservation intends to protect the data from unauthorized deletion, modification or fabrication. It is achieved using database constraints and transactions accomplished by a database management system. When the private data of the users are stored in the Cloud, the confidentiality of these data becomes essential to increase the reliability of the Cloud. Data confidentiality can be provided by authentication and access control policies, data encryption, and data storage distribution [Avizienis et al., 2004].

- Lock-in: Lock-in is one of the principal constraints of the Cloud environment. It concerns the cost of services mobility between different Clouds [Willcocks et al., 2013]. The difficulty of extracting and moving the data and the services from a Cloud to another one is preventing some organizations from adopting Cloud computing. This risk should be considered when subscribing to the Cloud services. In fact, there are no APIs for the data and processes in the Cloud computing, which limits the portability of the data and applications between different Clouds. Thus, if a company wants to change its Cloud supplier or if this latter goes bankrupt, the transfer of the data of the company from the current Cloud to another one will be a complex task and it will require a significant fee [Leavitt, 2009].

- Platform control: Businesses are generally wary of the Cloud environment. In fact, unlike the Cloud providers who can concept and change their platforms when and how they want and without the consent of the customers, companies are unable to change the technology of their platforms when they need. In addition, the conception of the Cloud platforms does not depend on business-specific IT and business practices, which will limit the appeal of the Cloud computing.

- Price: The cost dimension is based on several factors. It is defined according to the service provided, the rental period of the service, the quality of the service, the cost of maintenance, the age of the resources and the investment cost of the service provider [Al-Roomi et al., 2013].

- Service responsiveness: It consists of a graphical interface provided by the provider of Cloud computing to users. It helps manage issues and demands raised by Cloud users. This criterion includes the support reactiveness, the processing time of the customer's request and the acknowledgment time of the customer request [SLA, 2014].

- Traffic: This criterion is very important for real-time applications. Traditional real-time data processing systems assume that the network is predictable. They assume that the processing time of the data and the delivery time of the results are predictable [García-Valls et al., 2014].

- Service monitoring, verification and validation: This feature consists of monitoring and managing the processes within a cloud infrastructure. The goal is to ensure that the Cloud services are performing optimally. It is the use of manual or automated IT monitoring and management techniques to ensure that a cloud infrastructure or platform performs optimally.

- Business domain: This criterion refers to companies adopting the Cloud computing solution. It is the strategy used by the company to manage its resources and to offer the market the best services and products that its competitors. A good business model of an enterprise is supposed to define the customers and their needs and appreciated services, as well as the strategy to be adopted to earn money while offering the customers the services and products at appropriate costs [Afuah and Tucci, 2001].

- Data management: The availability of the data stored in the Cloud represents the degree to which the data can be easily and quickly used by the customers. Data must be available to the clients, on demand, even if the Cloud is down, as is the case during a network failure.

- Disaster recovery: Cloud disaster recovery is important features of the Cloud computing environment. It allows to backup and to restore the data of the users in disaster cases. It consists of storing several electronic copies of the users' data stored in the Cloud.

- Data localization: Data localization consists of storing the data on a device which is physically present within the borders of the country where the data was generated.

7.3.2.3 Potential network topologies

Three Information Technologies (ITs) infrastructures can be used to execute the data streams native filters solution: local computing architecture, Cloud computing architecture, and Fog computing architecture.

- **Network 1: Local computing:** With this Information Technology (IT) model, the computing are executed on local servers and the data are stored in the traditional data centers. In this case, the stored data and applications are accessed by the authorized users via a remote server.

- **Network 2: Cloud computing:** With this network architecture, two choices can be adopted: The single Cloud and the multi-Clouds. In the single Cloud architecture, an organization adopting the Cloud computing environment uses a single Cloud service provider to serve all its needs. This Cloud can be public, private, hybrid, etc. With this topology, the organization can vary the magnitude of the resources rented on the Cloud (server limit, number of

servers, etc.) according to its needs. The multi-Clouds architecture allows an organization to use different Clouds from different service providers for its projects. These Clouds can be of different types (public, private, hybrid, etc.). According to a recent Microsoft study, 79% [Hos, 2016] of the organizations using the Cloud environment adopt this network architecture.

- **Network 3: Fog computing:** With such topology, the data is forwarded to a fog computing server. Fog computing is a distributed infrastructure in which certain application processes or services are managed at the edge of the network by a smart device, but others are managed in the Cloud computing. It represents a layer between the Cloud and the hardware. Its primary goal is to reduce the amount of data which needs to be transported, processed, and stored in the Cloud.

7.3.2.4 Decision

In this section, we compare, based on a literature review, the three potential network topologies introduced in Section 7.3.2.3 against the decision factors introduced in Section 7.3.2.2.

The data validation algorithms used in Network 1 can still be used in the other two networks. In all three networks, the algorithms can run on the client's servers to ensure its results but can be equally hosted at another Cloud Service Provider (CSP). Similarly, for the service verification, the monitoring algorithms can be implemented on the client's servers or a third-party CSP.

Network 1 does not provide responsive scalability and requires over-provisioning to provide high availability and scalability. Over-provisioning is very expensive and decreases the utilization rate and Return On Investment (ROI). Therefore, Network 1 is considered inferior to Network 2 and Network 3 's high availability, scalability and payment on demand.

Disaster recovery requires redundant locations which can be very expensive for most clients. Network 1 requires extra investment to ensure disaster recovery, while the remaining networks natively provide this feature.

Data availability is related to the network's downtime which in turn related to the networks' defense against cyber-attacks, hardware failures, human errors, etc. Extremely low downtime is very expensive to achieve in traditional technologies, thus Network 1 cannot provide data availability with limited investment. Networks 2 and 3 natively provide this feature. If the CSP's Service Level Agreements (SLAs) are not very tight, and multi-Cloud solutions (multiple CSP) can provide this feature with a reasonable investment.

Network 1 natively provides the data localization feature. Data localization is not provided by all Cloud technologies. They require special SLAs to be able to provide data localization. However, dedicated servers such as the Fog computing technology (Network 3) can enforce the data localization feature.

The data security and anonymity against third-party users is provided by Network 1 since the data streams communicate directly with the sampling and cleaning processes at the client's premises. Third-parties can only access the data streams

summaries through the interface provided by the client. Network 1 is thus considered to be most secure for data storage. In Network 2, the data streams are processed at the CSP, thus data anonymity is ensured except against CSP. The fog servers in Network 3 can anonymize the data streams summaries before being sent to the CSP. In this case, the data anonymity remains ensured.

Network 1 cannot handle unpredicted traffic without over-provisioning which is very expensive and remain limited against peaks in traffic. The other two networks natively support traffic peaks with very limited investment. Full Data control can only be achieved by Network 1. Although Cloud computing literature provides mechanisms to ensure data control, it is left for the reader to evaluate the feasibility of these mechanisms. Both networks 2 and 3 are compatible with Big Data Analytics Software-as-a-Service (BDaaS) provided by CSPs. In Network 1, the summary has to be accessed by the CSP or uploaded, and this creates redundant and unnecessary traffic.

This discussion is shown in Table 7.3.

7.4 Conclusion

In the first part of this chapter, we have presented the data streams native filters module which corresponds to the "Filtres natifs" module of the WAVES platform global architecture (cf. Figure 7.1). This module has been developed and integrated in the platform within the WAVES project. The purpose of this module is to filter the received data streams in a qualitative and quantitative manner in order to enhance the quality of the data and reduce its volume. The native filters module consists of two processes: qualitative filter and quantitative filter. Upon receipt of the data from several streams continuously, the qualitative filter sub-module evaluates the quality of the data in terms of accuracy, completeness, and sensor reliability, and enhances the quality of the data in terms of accuracy and completeness by detecting, removing and replacing outliers and estimating missing data. After that, it adds several quality statistics to the data. Once qualified, the quantitative filter sub-module proceeds to summarize the data using sampling algorithms.

In the second part of this chapter, we studied the computational resources of the sampling and qualification processes, mainly the execution time and the storage space requirements are studied, and several computing servers and data storage models are considered. In this chapter, we also discussed the migration of the data streams native filters solution to the Cloud computing. The migration decision depends on several factors as discussed. Notice that the sampling process and cleaning process can be executed each on a different architecture. Actually, different options can be chosen to execute these processes: local computing and Cloud storage, Cloud computing and Cloud storage, local computing and local storage, and finally, Cloud computing and local storage.

TABLE 7.3 – Decision table based on the Cloud computing criteria.

Network topology / Criteria	Network 1	Network 2	Network 3
Data verification Data validation Data monitoring	Yes	Yes	Yes
Service verification Service validation Service monitoring	Yes	Yes	Yes
Service responsiveness Scalable	No	Yes	Yes
Disaster recovery	No	Yes	Yes
Data availability	No	Yes	Yes
Data localization	Yes	Can be	Can be
Data anonymity	Yes	No	Yes
Traffic predictability	No	Yes	Yes
Full data control	Yes	Can be	Can be
Price	N/A	N/A	N/A
BDaaS compatible?	No	Yes	Yes

Conclusion and Perspectives

The contributions of this thesis are multiple. Our goal in this thesis was to treat the chain, from the data collection to the data analysis of the streams generated by sensors. We have proposed the data streams native filtering as a solution to overcome the two problems related to data collection: the huge volume of the data generated by the sensors, and their poor quality. In the data analysis phase, we were interested in detecting the anomalies in the data.

In detail, we introduced in Chapter 1, the basic concepts of data streams, windowing models, and application domains. We detailed the different sampling algorithms used to construct a data stream summary and we focus particularly on their drawbacks.

We studied in Chapter 2, in depth, the sampling algorithm Chain-sample. The purpose of this algorithm is to select, randomly, at any time, a fixed proportion among the most recent elements of the stream contained in the last sliding window. Through a series of experiments, we have shown that this algorithm has several drawbacks essentially due to the collision and redundancy problems. To overcome these weaknesses, we proposed two approaches, the first is called "Inverting the selection for high sampling rates" and the second is inspired by the "Divide-to-Conquer" strategy. Various experiments have been carried out to show the effectiveness of these two improvements, in particular, their impact on the execution time of the algorithm.

We discussed in Chapter 3 the impact of the sampling process on the events detection. In the first part of this chapter, we discussed three data streams sampling algorithms. Their purpose is to keep a representative, fixed-size sample of the most recent elements of the stream over time. At first, the SRS and Deterministic sampling algorithms were adapted to the context of the sliding window. Secondly, the performance of these algorithms has been compared to that of the Chain-sample algorithm. The results of the experiments showed that Chain-sample gives better results than SRS and Deterministic algorithms in terms of execution time and sampling accuracy. The second part of this chapter was devoted to studying the impact of data sampling on anomaly detection results using EWMA algorithm. First, we developed a new version of the Weighted Random Sampling algorithm (WRS) that

samples the data based on their values with respect to the values of their neighbors in the current sliding window. Then, we sampled the data using the three algorithms: Deterministic sampling, SRS, and WRS. The obtained results showed that WRS algorithm performs better results than both Deterministic and SRS algorithms in terms of the true positive rate.

We illustrated in Chapter 4 the data quality aspects of sensor networks. We introduced the basic concepts of data quality. Then, we discussed the existing research studies related to data quality in sensor networks. Finally, we proposed a complete system for managing the quality of sensors data. This system addresses the evaluation and the improvement of data quality. For this purpose, four data quality dimensions are considered: precision, accuracy, confidence, and completeness.

We investigated in Chapter 5 the change detection in the time series emitted by sensors. In particular, we discussed the slow and gradual changes as they illustrate deviations in the sensor's calibration. We presented an in-depth analysis of CUSUM algorithm since it is well adapted to the detection of small deviations, and we proposed an improvement that provides a more precise description of the detected anomalies. First, we discussed the choice of CUSUM parameters in order to optimize its results. Secondly, we presented an efficient method for estimating the start and end times of the deviation detected by CUSUM. Finally, we adapted CUSUM to detect a deviation of the process variability. We applied this new version to the detection of the so-called stuck-at errors. All these improvements have been validated by simulation and against real data streams.

We focused in Chapter 6 on the spatial anomalies detection methods, more particularly, Moran scatterplot, a data visualization technique for graphically isolating spatial anomalies. This method calculates the distance between the points with respect to the considered observable while taking into account their spatial correlation degree (which closely depends on their geographical distance). At first, we proposed a new version of Moran scatterplot, in which we improved the calculation of the matrix of weights. This matrix is involved in the calculation of the distance. We calculated the weights based on several parameters that qualify the correlation between the spatial points. For this purpose, we used a robust distance metric called the Gower's coefficient to explore and to characterize the neighborhood of the spatial points. Then we applied the new version of Moran scatterplot to a new use case: the Parisian bicycle sharing system (Velib). Indeed, in this new context, there is clearly a strong spatial correlation since bikes availability in a given station is often similar to its neighboring stations. In this context, we defined the spatial anomaly as the station that is significantly different from its neighborhood: it is almost empty or full while its neighboring stations are globally balanced. The identification of these outlier stations allowed us to propose and then to evaluate a new method that encourages Velib users to better distribute the bicycles among the stations. We showed that this proposed method improves significantly resources availability, and consequently, the users satisfaction.

We presented in Chapter 7 the native filters module that we developed and integrated into the project WAVES platform. This module provides two features: real-time data qualification and data summarization. It is composed of two sub-modules: qualitative filter and quantitative filter. Upon receipt of the data, the qualitative filter evaluates and improves the quality of the data based on the ar-

chitecture presented in Chapter 4. Once qualified, the data will be summarized by the quantitative filter which in turn implements several sampling algorithms. We also evaluated in Chapter 7 the effectiveness of the native filters module according to the required computing resources. We presented a benchmark of the computing resources requirements for this solution while considering different infrastructures (local server vs. cloud server for data processing, and database vs. Hadoop for data storage). The computing resources cover the processing time of the input data and the amount of memory required to store the output data.

Many research axes related to data quality, data sampling, and anomalies detection issues have been identified and will be explored in our future work. Our future perspectives are summarized as follows:

a. **Adaptive sampling:** In the sampling algorithms considered in this thesis, the sampling rate is constant during the execution of the algorithm. In some use cases, it is useful to adapt dynamically the sampling rate to some varying conditions such as the available computational resources as proposed in [Schinkel and Chen, 2006]. We want to propose a sampling algorithm that adapts the sampling rate to data variance. The significant change in data variance can be detected using CUSUM as explained in Chapter 5 or another change detection method. For instance, in the case of the water distribution network supervision, the data related to the water consumption of the users in a residential area have very small variability during the night. It would be pertinent in that case to sample these data with a very low sampling rate. On the contrary, the water consumption data during the day may require a higher sampling rate given their high variability.

b. **Missing data recovery:** The loss of data has serious consequences for the environmental monitoring systems and can engender missing important events. In the water network monitoring case, the information recorded by the sensors represents the instantaneous rate of the delivered water. The loss of some data may lead to erroneous and distorted results about water consumption. That is why it is important to recover missed data, when possible. In the native filters module that we developed, we replaced missed data using temporal interpolation on a single time series. One perspective would be to explore other approaches such as exploiting periodicity in the time series. Spatial correlation between data sensors can be also exploited to recover missed data.

c. **Precision data quality dimension:** As we presented in Chapter 4, the precision degree of the data is represented by the noise level in these data. This degree is calculated according to the Coefficient of Variation denoted by CV. According to the CV value and to the predefined thresholds, the data precision can be considered as Good, Medium, or Bad. When the data precision is Medium, the data have to be denoised using smoothing algorithms. One perspective would be to compare several smoothing techniques such as Moving Average, Median Average, and Savitzky-Golay algorithms, define the CV thresholds, and finally, implement these features in the native filters module.

d. **Anomalies detection:** In this thesis, we detected anomalies in two different application domains: water distribution network and bike sharing systems. In

both cases, the performed analysis is based on dynamic and static data collected from these systems. In our future work, we will enrich this kind of data with other sources of information such as social networks or weather conditions. Correlating information collected from different heterogeneous sources leads to the inference of new knowledge. It enables to predict the future evolution of the system with a good precision and to improve the reactivity of anomaly detection process. In this context, semantic web technologies can be used to deal with the heterogeneity of data sources.

e. **Distribution of service:** We studied in Chapter **??** the execution of data streams native filters module in the context of the cloud. Indeed, Cloud computing provides efficiency, flexibility and cost savings. Traditionally, data analysis tasks are conducted separately by different organizations. However, these tasks include many common steps such as information retrieval, data cleaning, data visualization and data storage. Building separate systems to analyze the data is very expensive and requires specific skills.

   Big Data Analytics Software-as-a-Service is an emerging category of services for massive data processing in the Cloud computing. The offered services are the traditional data analysis tasks. These tasks are grouped and provided to the customer for a single fee and upon request [Zheng et al., 2013]. With the use of Big Data Analytics Software-as-a-Service, the cost of storage drops significantly and the efforts to install, configure and maintain the data analysis systems are saved. Moreover, the estimation and the planning of the required resources are no longer necessary as the resources can be flexibly adjusted as needed. For instance, Google BigQuery [Goo, 2011] is an example of such a platform that performs real-time Big Data analysis in the cloud environment. We can also mention Spark, an Apache project for streaming processing in the cloud, and Elastic MapReduce (Hadoop) for Big data analytics in the cloud [Hashem et al., 2015]. We are orienting our future work towards the use of such solutions.

   The Cloud computing environment presents several challenges, particularly, the data security. In order to ensure the reliability of the cloud and the trust of the users regarding this environment, it is necessary to protect and to ensure the security of the data stored in the cloud. Our future work tends to focus on this subject.

References

[TRA, 1999] (1999). https://traderbotmarketplace.com.

[TRU, 2004] (2004). http://www.truviso.com/.

[lib, 2010] (2010). Detecting forest fires using wireless sensor networks. http://www.libelium.com/wireless_sensor_networks_to_detec_forest_fires/.

[Goo, 2011] (2011). Google bigquery. https://cloud.google.com/bigquery/?hl=fr.

[vel, 2014] (2014). Tout sur Vélib. http://blog.velib.paris.fr/blog/2014/07/15/7-ans-de-velib-des-records-de-frequentation-et-dabonnements/.

[Hos, 2016] (2016). Cloud and Hosting Trends - The Digital Revolution, Powered by Cloud. https://www.microsoft.com/en-us/download/details.aspx?id=52045.

[Abadi et al., 2005] Abadi, D. J., Ahmad, Y., Balazinska, M., Cetintemel, U., Cherniack, M., Hwang, J.-H., Lindner, W., Maskey, A., Rasin, A., Ryvkina, E., et al. (2005). The design of the borealis stream processing engine. In *Cidr*, volume 5, pages 277–289.

[Abadi et al., 2003] Abadi, D. J., Carney, D., Çetintemel, U., Cherniack, M., Convey, C., Lee, S., Stonebraker, M., Tatbul, N., and Zdonik, S. (2003). Aurora: a new model and architecture for data stream management. *the VLDB Journal*, 12(2):120–139.

[Abdessalem et al., 2007] Abdessalem, T., Chiky, R., Hébrail, G., Vitti, J. L., and Paris, G.-E. (2007). Traitement de données de consommation électrique par un système de gestion de flux de données. In *EGC*, pages 521–532.

[Abuaitah and Wang, 2015] Abuaitah, G. R. and Wang, B. (2015). A taxonomy of sensor network anomalies and their detection approaches. In *Technological Breakthroughs in Modern Wireless Sensor Applications*, pages 172–206. IGI Global.

[Afuah and Tucci, 2001] Afuah, A. and Tucci, C. L. (2001). *Internet business models and strategies*. McGraw-Hill New York.

[Akhtar, 2011] Akhtar, N. (2011). Statistical data analysis of continuous streams using stream dsms. *International Journal of Database Management Systems (IJDMS)*, 3(2):89–99.

[Al-Roomi et al., 2013] Al-Roomi, M., Al-Ebrahim, S., Buqrais, S., and Ahmad, I. (2013). Cloud computing pricing models: a survey. *International Journal of Grid and Distributed Computing*, 6(5):93–106.

[Anselin, 1993] Anselin, L. (1993). *The Moran scatterplot as an ESDA tool to assess local instability in spatial association*. Regional Research Institute, West Virginia University Morgantown, WV.

[Arasu et al., 2016] Arasu, A., Babcock, B., Babu, S., Cieslewicz, J., Datar, M., Ito, K., Motwani, R., Srivastava, U., and Widom, J. (2016). Stream: The stanford data stream management system. In *Data Stream Management*, pages 317–336. Springer.

[Armbrust et al., 2010] Armbrust, M., Fox, A., Griffith, R., Joseph, A. D., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I., et al. (2010). A view of cloud computing. *Communications of the ACM*, 53(4):50–58.

[Avizienis et al., 2004] Avizienis, A., Laprie, J.-C., Randell, B., and Landwehr, C. (2004). Basic concepts and taxonomy of dependable and secure computing. *IEEE transactions on dependable and secure computing*, 1(1):11–33.

[Babcock et al., 2002] Babcock, B., Datar, M., and Motwani, R. (2002). *Sampling from a moving window over streaming data*. In *Proceedings of the thirteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 633–634.

[Babu et al., 2001] Babu, S., Subramanian, L., and Widom, J. (2001). A data stream management system for network traffic management. In *In Proceedings of Workshop on Network-Related Data Management (NRDM 2001)*.

[Bartok et al., 2010] Bartok, J., Habala, O., Bednar, P., Gazak, M., and Hluchỳ, L. (2010). Data mining and integration for predicting significant meteorological phenomena. *Procedia Computer Science*, 1(1):37–46.

[Basseville et al., 1993] Basseville, M., Nikiforov, I. V., et al. (1993). *Detection of abrupt changes: theory and application*, volume 104. Prentice Hall Englewood Cliffs.

[Batini and Scannapieco, 2010] Batini, C. and Scannapieco, M. (2010). *Data Quality: concepts, methodologies and techniques*. Springer-Verlag Berlin Heidelberg.

[Belkin, 1997] Belkin, L. (1997). How can we save the next victim?

[Bettencourt et al., 2007] Bettencourt, L. M., Hagberg, A. A., and Larkey, L. B. (2007). Separating the wheat from the chaff: Practical anomaly detection schemes in ecological applications of distributed sensor networks. In *International Conference on Distributed Computing in Sensor Systems*, pages 223–239. Springer.

[Bissell, 1969] Bissell, A. (1969). Cusum techniques for quality control. *Applied Statistics*, pages 1–30.

[Blumberg and Atre, 2003] Blumberg, R. and Atre, S. (2003). The problem with unstructured data. *Dm Review*, 13(42-49):62.

[Borgnat et al., 2011] Borgnat, P., Abry, P., Flandrin, P., Robardet, C., Rouquier, J.-B., and Fleury, E. (2011). Shared bicycles in a city: A signal processing and data analysis perspective. *Advances in Complex Systems*, 14(03):415–438.

[Bošnjak and Cisar, 2010] Bošnjak, S. and Cisar, S. M. (2010). Ewma based threshold algorithm for intrusion detection. *Computing and Informatics*, 29:1089–1101.

[Brauckhoff et al., 2006] Brauckhoff, D., Tellenbach, B., Wagner, A., May, M., and Lakhina, A. (2006). Impact of packet sampling on anomaly detection metrics. In *Proceedings of the 6th ACM SIGCOMM conference on Internet measurement*, pages 159–164. ACM.

[Brettlecker and Schuldt, 2007] Brettlecker, G. and Schuldt, H. (2007). The osiris-se (stream-enabled) infrastructure for reliable data stream management on mobile devices. In *Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, pages 1097–1099. ACM.

[Brook and Evans, 1972] Brook, D. and Evans, D. (1972). *An approach to the probability distribution of CUSUM run length*. *Biometrika*, pages 539–549.

[Carney et al., 2002] Carney, D., Cetinternel, U., Cherniack, M., Convey, C., Lee, S., Seidman, G., Stonebraker, M., Tatbul, N., and Zdonik, S. (2002). Monitoring streams–a new class of dbms applications. Technical report, Technical Report CS-02-01, Department of Computer Science, Brown University.

[Cetintemel, 2003] Cetintemel, U. (2003). The aurora and medusa projects. *Data Engineering*, 51(3).

[Chabchoub and C., 2014] Chabchoub, Y. and C., F. (2014). Analyse des trajets de vélib par clustering. *Extraction et gestion des connaissances, Clustering and Co-clustering*.

[Chabchoub et al., 2014] Chabchoub, Y., Chiky, R., and Dogan, B. (2014). *How can sliding HyperLogLog and EWMA detect port scan attacks in IP traffic? EURASIP Journal on Information Security*, 2014(1):5.

[Chabchoub and Fricker, 2014] Chabchoub, Y. and Fricker, C. (2014). Classification of the vélib stations using kmeans, dynamic time wraping and dba averaging method. In *Computational Intelligence for Multimedia Understanding (IWCIM), 2014 International Workshop on*, pages 1–5. IEEE.

[Chakravarthy and Jiang, 2009] Chakravarthy, S. and Jiang, Q. (2009). *Stream data processing: a quality of service perspective: modeling, scheduling, load shedding, and complex event processing*, volume 36. Springer Science & Business Media.

[Chandola et al., 2009] Chandola, V., Banerjee, A., and Kumar, V. (2009). Anomaly detection: A survey. *ACM computing surveys (CSUR)*, 41(3):15.

[Chandrasekaran et al., 2003] Chandrasekaran, S., Cooper, O., Deshpande, A., Franklin, M. J., Hellerstein, J. M., Hong, W., Krishnamurthy, S., Madden, S. R., Reiss, F., and Shah, M. A. (2003). Telegraphcq: continuous dataflow processing. In *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, pages 668–668. ACM.

[Chao, 1982] Chao, M. (1982). *A general purpose unequal probability sampling plan*. *Biometrika*, 69(3):653–656.

[Chen et al., 2005] Chen, D., Shao, X., Hu, B., and Su, Q. (2005). Simultaneous wavelength selection and outlier detection in multivariate regression of near-infrared spectra. *Analytical Sciences*, 21(2):161–166.

[Chen et al., 2000] Chen, J., DeWitt, D. J., Tian, F., and Wang, Y. (2000). Niagaracq: A scalable continuous query system for internet databases. In *ACM SIGMOD Record*, volume 29, pages 379–390. ACM.

[Cheng et al., 2018] Cheng, H., Feng, D., Shi, X., and Chen, C. (2018). Data quality analysis and cleaning strategy for wireless sensor networks. *EURASIP Journal on Wireless Communications and Networking*, page 61.

[Čisar and Čisar, 2011] Čisar, P. and Čisar, S. M. (2011). Optimization methods of ewma statistics. *Acta Polytechnica Hungarica*, 8(5):73–87.

[Cochran, 1977] Cochran, W. (1977). *Sampling Techniques*. John Wiley & Sons.

[Cool, 2000] Cool, A. L. (2000). A review of methods for dealing with missing data.

[Cormode, 2013] Cormode, G. (2013). The continuous distributed monitoring model. *ACM SIGMOD Record*, 42(1):5–14.

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

[Cranor et al., 2002] Cranor, C., Gao, Y., Johnson, T., Shkapenyuk, V., and Spatscheck, O. (2002). Gigascope: High performance network monitoring with an sql interface. In *Proceedings of the 2002 ACM SIGMOD international conference on Management of data*, pages 623–623. ACM.

[Crovella and Bestavros, 1997] Crovella, M. E. and Bestavros, A. (1997). Self-similarity in world wide web traffic: evidence and possible causes. *IEEE/ACM Transactions on networking*, 5(6):835–846.

[Csernel, 2008] Csernel, B. (2008). *Résumé généraliste de flux de données*. PhD thesis, Paris, ENST.

[Csernel et al., 2006] Csernel, B., Clerot, F., and Hébrail, G. (2006). *Datastream clustering over tilted windows through sampling*. *Knowledge Discovery from Data Streams*, page 127.

[Dash and Ng, 2006] Dash, M. and Ng, W. (2006). *Efficient reservoir sampling for transactional data streams*. In *Sixth IEEE International Conference on Data Mining-Workshops (ICDMW'06)*, pages 662–666. IEEE.

[de Aquino et al., 2007] de Aquino, A. L., Figueiredo, C. M. S., Nakamura, E. F., Buriol, L. S., Loureiro, A. A. F., Fernandes, A. O., and Coelho Jr, C. J. N. (2007). A sampling data stream algorithm for wireless sensor networks. In *Communications, 2007. ICC'07. IEEE International Conference on*, pages 3207–3212. IEEE.

[Dhariwal et al., 2006] Dhariwal, A., de Menezes Pereira, A. A., Oberg, C., Stauffer, B., Moorthi, S., Caron, D. A., Sukhatme, G., et al. (2006). Namos: Networked aquatic microbial observing system.

[Dunning and Friedman, 2014] Dunning, T. and Friedman, E. (2014). *Practical machine learning: a new look at anomaly detection*. " O'Reilly Media, Inc.".

[Efraimidis, 2015] Efraimidis, P. S. (2015). *Weighted random sampling over data streams*. In *Algorithms, Probability, Networks, and Games*, pages 183–195. Springer.

[Efraimidis and Spirakis, 2006] Efraimidis, P. S. and Spirakis, P. G. (2006). *Weighted random sampling with a reservoir*. *Information Processing Letters*, pages 181–185.

[Elnahrawy and Nath, 2003] Elnahrawy, E. and Nath, B. (2003). Cleaning and querying noisy sensors. In *Proceedings of the 2nd ACM international conference on Wireless sensor networks and applications*, pages 78–87. ACM.

[Etienne and Latifa, 2014] Etienne, C. and Latifa, O. (2014). Model-based count series clustering for bike sharing system usage mining: a case study with the vélib'system of paris. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 5(3):39.

[Ewan, 1963] Ewan, W. D. (1963). When and how to use cu-sum charts. *Technometrics*, 5(1):1–22.

[Fan, 2015] Fan, W. (2015). *Data Quality: From Theory to Practice*. *ACM SIGMOD Record*, 44(3):7–18.

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

[Fricker and Gast, 2014] Fricker, C. and Gast, N. (2014). Incentives and regulations in bike-sharing systems with stations of finite capacity, special issue: Shared mobility systems. *EURO Journal on Transportation and Logistics*.

[Froehlich et al., 2009] Froehlich, J., Neumann, J., and Oliver, N. (2009). Sensing and predicting the pulse of the city through shared bicycling. In *Proceedings of the 21st International Jont Conference on Artifical Intelligence*, IJCAI'09, pages 1420–1426, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

[Gabsi, 2011] Gabsi, N. (2011). *Extension et interrogation de résumés de flux de données*. PhD thesis, Télécom ParisTech.

[Galeano et al., 2006] Galeano, P., Peña, D., and Tsay, R. S. (2006). Outlier detection in multivariate time series by projection pursuit. *Journal of the American Statistical Association*, 101(474):654–669.

[García-Valls et al., 2014] García-Valls, M., Cucinotta, T., and Lu, C. (2014). Challenges in real-time virtualization and predictable cloud computing. *Journal of Systems Architecture*, 60(9):726–740.

[Gemulla, 2008] Gemulla, R. (2008). *Sampling algorithms for evolving datasets*. PhD thesis, Technischen Universitat Dresden Fakultat Informatik.

[Gemulla et al., 2006] Gemulla, R., Lehner, W., and Haas, P. J. (2006). *A dip in the reservoir: Maintaining sample synopses of evolving datasets*. In *Proceedings of the 32nd international conference on Very large data bases*, pages 595–606. VLDB Endowment.

[Gibbons and Matias, 1998] Gibbons, P. B. and Matias, Y. (1998). *New sampling-based summary statistics for improving approximate query answers*. In *ACM SIGMOD Record*, volume 27, pages 331–342. ACM.

[Gibbons et al., 1997] Gibbons, P. B., Matias, Y., and Poosala, V. (1997). *Fast incremental maintenance of approximate histograms*. In *VLDB*, volume 97, pages 466–475.

[Gibbons et al., 2002] Gibbons, P. B., Matias, Y., and Poosala, V. (2002). *Fast incremental maintenance of approximate histograms*. *ACM Transactions on Database Systems (TODS)*, pages 261–298.

[Gietl and Klemm, 2009] Gietl, J. K. and Klemm, O. (2009). Analysis of traffic and meteorology on airborne particulate matter in münster, northwest germany. *Journal of the Air & Waste Management Association*, 59(7):809–818.

[Gilbert et al., 2001] Gilbert, A. C., Kotidis, Y., Muthukrishnan, S., and Strauss, M. (2001). Quicksand: Quick summary and analysis of network data. Technical report, Technical Report, Dec. 2001. citeseer. nj. nec. com/gilbert01quicksand. html.

[Goel and Wu, 1973] Goel, A. L. and Wu, S. (1973). Economically optimum design of cusum charts. *Management Science*, 19(11):1271–1282.

[Golab and Özsu, 2003a] Golab, L. and Özsu, M. T. (2003a). Data stream management issues–a survey. Technical report, Technical Report, Apr. 2003. db. uwaterloo. ca/~ ddbms/publications/stream/streamsurvey. pdf.

[Golab and Özsu, 2003b] Golab, L. and Özsu, M. T. (2003b). Issues in data stream management. *ACM Sigmod Record*, 32(2):5–14.

[Gower, 1971] Gower, J. C. (1971). A general coefficient of similarity and some of its properties. *Biometrics*, pages 857–871.

[Gravetter and Forzano, 2018] Gravetter, F. J. and Forzano, L.-A. B. (2018). *Research methods for the behavioral sciences*. Cengage Learning.

[Gruenwald et al., 2007] Gruenwald, L., Chok, H., and Aboukhamis, M. (2007). Using data mining to estimate missing sensor data. In *Seventh IEEE International Conference on Data Mining Workshops (ICDMW 2007)*, pages 207–212. IEEE.

[Gürgen, 2007] Gürgen, L. (2007). *Gestion à grande échelle de données de capteurs hétérogènes*. PhD thesis, Grenoble, INPG.

[Gutiérrez Rodriguez, 2010] Gutiérrez Rodriguez, C. (2010). *Qualité des données capteurs pour les systèmes de surveillance de phénomènes environnementaux*. PhD thesis, Villeurbanne, INSA.

[Haining, 1993] Haining, R. (1993). *Spatial data analysis in the social and environmental sciences*. Cambridge University Press.

[Hashem et al., 2015] Hashem, I. A. T., Yaqoob, I., Anuar, N. B., Mokhtar, S., Gani, A., and Khan, S. U. (2015). The rise of âĂIJbig dataâĂİ on cloud computing: Review and open research issues. *Information Systems*, 47:98–115.

[Haslett et al., 1991] Haslett, J., Bradley, R., Craig, P., Unwin, A., and Wills, G. (1991). Dynamic graphics for exploring spatial data with application to locating global and local anomalies. *The American Statistician*, 45(3):234–242.

[Hermans, 2009] Hermans, F. (2009). *Data Fusion Based on Distributed Quality Estimation in Wireless Sensor Networks*. PhD thesis, Fachbereich Mathematik und Informatik Institut fur Informatik.

[Hermans et al., 2009] Hermans, F., Dziengel, N., and Schiller, J. (2009). *Quality estimation based data fusion in wireless sensor networks*. In *Mobile Adhoc and Sensor Systems, 2009. MASS'09. IEEE 6th International Conference on*, pages 1068–1070. IEEE.

[Iannacchione, 1982] Iannacchione, V. G. (1982). Weighted sequential hot deck imputation macros. In *Seventh Annual SAS UserâĂŹs Group International Conference, San Francisco*.

[Ingelrest et al., 2010] Ingelrest, F., Barrenetxea, G., Schaefer, G., Vetterli, M., Couach, O., and Parlange, M. (2010). Sensorscope: Application-specific sensor network for environmental monitoring. *ACM Transactions on Sensor Networks (TOSN)*, 6(2):17.

[Islam et al., 2014] Islam, M. Z., Mamun, Q., and Rahman, M. G. (2014). Data cleansing during data collection from wireless sensor networks.

[Jain and Chang, 2004] Jain, A. and Chang, E. Y. (2004). Adaptive sampling for sensor networks. In *Proceeedings of the 1st international workshop on Data management for sensor networks: in conjunction with VLDB 2004*, pages 10–16. ACM.

[Jarke et al., 1999] Jarke, M., Jeusfeld, M. A., Quix, C., and Vassiliadis, P. (1999). *Architecture and quality in data warehouses: An extended repository approach*. *Information Systems*, 24(3):229–253.

[Jeffery et al., 2006] Jeffery, S. R., Alonso, G., Franklin, M. J., Hong, W., and Widom, J. (2006). *Declarative support for sensor data cleaning*. In *Pervasive Computing*, pages 83–100. Springer.

[Kaiser et al., 2005] Kaiser, W. J., Pottie, G. J., Srivastava, M., Sukhatme, G. S., Villasenor, J., and Estrin, D. (2005). Networked infomechanical systems (nims) for ambient intelligence. In *Ambient Intelligence*, pages 83–113. Springer.

[Karr et al., 2006] Karr, A. F., Sanil, A. P., and Banks, D. L. (2006). Data quality: A statistical perspective. *Statistical Methodology*, 3(2):137–173.

[Klein et al., 2007] Klein, A., Do, H.-H., Hackenbroich, G., Karnstedt, M., and Lehner, W. (2007). *Representing data quality for streaming and static data*. In *Data Engineering Workshop, 2007 IEEE 23rd International Conference on*, pages 3–10.

[Klein and Lehner, 2009] Klein, A. and Lehner, W. (2009). *Representing data quality in sensor data streaming environments*. *Journal of Data and Information Quality (JDIQ)*.

[Knorn and Leith, 2008] Knorn, F. and Leith, D. J. (2008). Adaptive kalman filtering for anomaly detection in software appliances. In *INFOCOM Workshops 2008, IEEE*, pages 1–6. IEEE.

[Kovalerchuk and Vityaev, 2000] Kovalerchuk, B. and Vityaev, E. (2000). *Data mining in finance: advances in relational and hybrid methods*, volume 547. Springer Science & Business Media.

[Küçük et al., 2015] Küçük, D., İnan, T., Boyrazoğlu, B., Buhan, S., Salor, Ö., Çadırcı, I., and Ermiş, M. (2015). Pqstream: A data stream architecture for electrical power quality. *arXiv preprint arXiv:1504.04750*.

[Laird et al., 2001] Laird, T., Patton, R., and Ogle, J. (2001). *Designing SQL Server 2000 Databases*. Syngress Publishing.

[Le Gruenwald, 2005] Le Gruenwald, M. H. (2005). Estimating missing values in related sensor data streams. In *COMAD*.

[Leavitt, 2009] Leavitt, N. (2009). Is cloud computing really ready for prime time. *Growth*, 27(5):15–20.

[Lei et al., 2016] Lei, J., Bi, H., Xia, Y., Huang, J., and Bae, H. (2016). An in-network data cleaning approach for wireless sensor networks. *Intelligent Automation & Soft Computing*, 22(4):599–604.

[Leland et al., 1993] Leland, W. E., Taqqu, M. S., Willinger, W., and Wilson, D. V. (1993). On the self-similar nature of ethernet traffic. In *ACM SIGCOMM computer communication review*, volume 23, pages 183–193. ACM.

[Li et al., 2012] Li, F., Nastic, S., and Dustdar, S. (2012). Data quality observation in pervasive environments. In *Computational Science and Engineering (CSE), 2012 IEEE 15th International Conference on*, pages 602–609. IEEE.

[Li and Parker, 2008] Li, Y. and Parker, L. E. (2008). Classification with missing data in a wireless sensor network. In *Southeastcon, 2008. IEEE*, pages 533–538. IEEE.

[Lim et al., 2009] Lim, H.-S., Moon, Y.-S., and Bertino, E. (2009). *Research issues in data provenance for streaming environments*. In *Proceedings of the 2nd SIGSPATIAL ACM GIS 2009 International Workshop on Security and Privacy in GIS and LBS*, pages 58–62.

[Little and Rubin, 2014] Little, R. J. and Rubin, D. B. (2014). *Statistical analysis with missing data*, volume 333. John Wiley & Sons.

[Liu et al., 2003] Liu, J., Liu, J., Reich, J., Cheung, P., and Zhao, F. (2003). *Distributed group management for track initiation and maintenance in target localization applications*. In *Information Processing in Sensor Networks*, pages 113–128. Springer.

[Liu and Chi, 2002] Liu, L. and Chi, L. (2002). *Evolutionary data quality*. In *Proceedings of the 7th International Conference on Information Quality*.

[Liu et al., 1999] Liu, L., Pu, C., and Tang, W. (1999). Continual queries for internet scale event-driven information delivery. *IEEE Transactions on Knowledge and Data Engineering*, 11(4):610–628.

[Liu et al., 2000] Liu, L., Pu, C., and Tang, W. (2000). Webcq-detecting and delivering information changes on the web. In *Proceedings of the ninth international conference on Information and knowledge management*, pages 512–519. ACM.

[Lohninger, 2010] Lohninger, H. (2010). Fundamentals of statistics. *Retrieved December*, 5:2010.

[Lu et al., 2003] Lu, C.-T., Chen, D., and Kou, Y. (2003). Algorithms for spatial outlier detection. In *Data Mining, 2003. ICDM 2003. Third IEEE International Conference on*, pages 597–600. IEEE.

[Lu and Liang, 2004] Lu, C.-T. and Liang, L. R. (2004). Wavelet fuzzy classification for detecting and tracking region outliers in meteorological data. In *Proceedings of the 12th annual ACM international workshop on Geographic information systems*, pages 258–265. ACM.

[Lucas and Crosier, 1982a] Lucas, J. M. and Crosier, R. B. (1982a). Fast initial response for cusum quality-control schemes: give your cusum a head start. *Technometrics*, 24(3):199–205.

[Lucas and Crosier, 1982b] Lucas, J. M. and Crosier, R. B. (1982b). *Fast Initial Response for CUSUM Quality-Control Schemes: Give Your CUSUM A Head Start* . In *Technometrics*, pages 199–205.

[Lucas and Saccucci, 1990] Lucas, J. M. and Saccucci, M. S. (1990). Exponentially weighted moving average control schemes: properties and enhancements. *Technometrics*, 32(1):1–12.

[Madnick et al., 2005] Madnick, S., Wang, R., Chettayar, K., Dravis, F., Funk, J., Katz-Haas, R., Lee, C., Lee, Y., Xian, X., and Bhansali, S. (2005). Exemplifying business opportunities for improving data quality from corporate household research. *Information Quality*, pages 181–196.

[Mai et al., 2006] Mai, J., Chuah, C.-N., Sridharan, A., Ye, T., and Zang, H. (2006). Is sampled data sufficient for anomaly detection? In *Proceedings of the 6th ACM SIGCOMM conference on Internet measurement*, pages 165–176. ACM.

[Manandhar et al., 2014] Manandhar, K., Cao, X., Hu, F., and Liu, Y. (2014). Detection of faults and attacks including false data injection attack in smart grid using kalman filter. *IEEE transactions on control of network systems*, 1(4):370–379.

[Manonmani and Suganya, 2010] Manonmani, R. and Suganya, G. M. D. (2010). *Remote sensing and GIS application in change detection study in urban zone using multi temporal satellite*. *International journal of Geomatics and Geosciences*, 1(1):60.

[Mathioudakis and Koudas, 2010] Mathioudakis, M. and Koudas, N. (2010). Twittermonitor: trend detection over the twitter stream. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, pages 1155–1158. ACM.

[McLachlan and Krishnan, 2007] McLachlan, G. and Krishnan, T. (2007). *The EM algorithm and extensions*, volume 382. John Wiley & Sons.

[McLeod and Bellhouse, 1983] McLeod, A. I. and Bellhouse, D. R. (1983). *A convenient algorithm for drawing a simple random sample*. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, pages 182–184.

[Mell et al., 2011] Mell, P., Grance, T., et al. (2011). The nist definition of cloud computing.

[Midas et al., 2010] Midas, C. D. et al. (2010). Résumé généraliste de flux de données. In *EGC: Extraction et Gestion des Connaissances*, pages 255–260.

[Moges, 2014] Moges, H. T. (2014). A contextual data quality analysis for credit risk management in financial institutions.

[Montgomery, 2007] Montgomery, D. C. (2007). *Introduction to statistical quality control*. John Wiley & Sons.

[Muthukrishnan et al., 2005] Muthukrishnan, S. et al. (2005). Data streams: Algorithms and applications. *Foundations and Trends® in Theoretical Computer Science*, 1(2):117–236.

[Nair, 2009] Nair, S. D. (2009). *Finding fault: Anomaly detection for embedded networked sensing*. University of California, Los Angeles.

[Naumann, 2002] Naumann, F. (2002). *Quality-driven query answering for integrated information systems*, volume 2261. Springer Science & Business Media.

[Neely, 2005] Neely, M. P. (2005). The product approach to data quality and fitness for use: a framework for analysis.

[Nepal et al., 2015] Nepal, S., Ranjan, R., and Choo, K.-K. R. (2015). Trustworthy processing of healthcare big data in hybrid clouds. *IEEE Cloud Computing*, 2(2):78–84.

[Ni et al., 2009] Ni, K., Ramanathan, N., Chehade, M. N. H., Balzano, L., Nair, S., Zahedi, S., Kohler, E., Pottie, G., Hansen, M., and Srivastava, M. (2009). Sensor network data fault types. *ACM Transactions on Sensor Networks (TOSN)*, 5(3):25.

[Olbrich, 2010] Olbrich, S. (2010). *Warehousing and Analyzing Streaming Data Quality Information*. In *AMCIS*, page 159.

[Olken and Rotem, 1992] Olken, F. and Rotem, D. (1992). *Maintenance of materialized views of sampling queries*. In *Data Engineering, 1992. Proceedings. Eighth International Conference on*, pages 632–641. IEEE.

[Osborne et al., 2014] Osborne, M., Moran, S., McCreadie, R., Von Lunen, A., Sykora, M. D., Cano, E., Ireson, N., Macdonald, C., Ounis, I., He, Y., et al. (2014). Real-time detection, tracking, and monitoring of automatically discovered events in social media.

[Page, 1954] Page, E. (1954). Continuous inspection schemes. *Biometrika*, 41(1/2):100–115.

[Pan et al., 2010] Pan, L., Li, J., et al. (2010). K-nearest neighbor based missing data estimation algorithm in wireless sensor networks. *Wireless Sensor Network*, 2(02):115.

[Pang et al., 2008] Pang, Q., Wong, V. W., and Leung, V. C. (2008). Reliable data transport and congestion control in wireless sensor networks. *International journal of sensor networks*, 3(1):16–24.

[Patton and Ogle, 2001] Patton, R. and Ogle, J. (2001). *Designing SQL Server 2000 Databases for. Net Enterprise Servers*. Syngress Press.

[Pena et al., 2013] Pena, E. H., de Assis, M. V., and Proença, M. L. (2013). Anomaly detection using forecasting methods arima and hwds. In *Chilean Computer Science Society (SCCC), 2013 32nd International Conference of the*, pages 63–66. IEEE.

[Peng et al., 2007] Peng, T., Leckie, C., and Ramamohanarao, K. (2007). Information sharing for distributed intrusion detection systems. *Journal of Network and Computer Applications*, 30(3):877–899.

[Pescapé et al., 2010] Pescapé, A., Rossi, D., Tammaro, D., and Valenti, S. (2010). On the impact of sampling on traffic monitoring and analysis. In *Teletraffic Congress (ITC), 2010 22nd International*, pages 1–8. IEEE.

[Provost et al., 1999] Provost, F., Jensen, D., and Oates, T. (1999). *Efficient progressive sampling*. In *Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 23–32. ACM.

[Rahman et al., 2012] Rahman, M. G., Islam, M. Z., Bossomaier, T., and Gao, J. (2012). Cairad: a co-appearance based analysis for incorrect records and attribute-values detection. In *Neural Networks (IJCNN), The 2012 International Joint Conference on*, pages 1–10. IEEE.

[Ramanathan et al., 2006] Ramanathan, N., Schoellhammer, T., Estrin, D., Hansen, M., Harmon, T., Kohler, E., and Srivastava, M. (2006). The final frontier: Embedding networked sensors in the soil. *Center for Embedded Network Sensing*.

[Ramirez et al., 2011] Ramirez, G., Fuentes, O., and Tweedie, C. E. (2011). *Assessing data quality in a sensor network for environmental monitoring*. In *Fuzzy Information Processing Society (NAFIPS), 2011 Annual Meeting of the North American*, pages 1–6. IEEE.

[Redman, 1997] Redman, T. C. (1997). *Data quality for the information age*. Artech House, Inc.

[Reynolds, 1975] Reynolds, M. R. (1975). Approximations to the average run length in cumulative sum control charts. *Technometrics*, 17(1):65–71.

[Reynolds and Stoumbos, 2010] Reynolds, M. R. and Stoumbos, Z. G. (2010). Robust cusum charts for monitoring the process mean and variance. *Quality and Reliability Engineering International*, 26(5):453–473.

[Roberts, 1959] Roberts, S. (1959). Control chart tests based on geometric moving averages. *Technometrics*, 1(3):239–250.

[Rodriguez and Servigne, 2012] Rodriguez, C. G. and Servigne, S. (2012). *Sensor Data Quality for Geospatial Monitoring Applications*. In *AGILE 2012, 15th Internationale Conference on Geographic Information Science*, pages 1–6.

[Rubin, 2004] Rubin, D. B. (2004). *Multiple imputation for nonresponse in surveys*, volume 81. John Wiley & Sons.

[Sachpazidis, 2002] Sachpazidis, J. (2002). @ home: A modular telemedicine system. In *Mobile Computing in Medicine, Second Conference on Mobile Computing in Medicine, Workshop of the Project Group MoCoMed, GMDS-Fachbereich Medizinische Informatik & GI-Fachausschuss 4.7*, pages 87–95. GI.

[Saleh et al., 2015] Saleh, N. A., Mahmoud, M. A., Jones-Farmer, L. A., Zwetsloot, I., and Woodall, W. H. (2015). Another look at the ewma control chart with estimated parameters. *Journal of Quality Technology*, 47(4):363–382.

[Schinkel and Chen, 2006] Schinkel, M. and Chen, W.-H. (2006). Control of sampled-data systems with variable sampling rate. *International journal of systems science*, 37(9):609–618.

[Sharma et al., 2010] Sharma, A. B., Golubchik, L., and Govindan, R. (2010). Sensor faults: Detection methods and prevalence in real-world datasets. *ACM Transactions on Sensor Networks (TOSN)*, 6(3):23.

[Shekhar et al., 2011] Shekhar, S., Evans, M. R., Kang, J. M., and Mohan, P. (2011). Identifying patterns in spatial information: A survey of methods. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 1(3):193–214.

[Shekhar et al., 2001] Shekhar, S., Lu, C.-T., and Zhang, P. (2001). Detecting graph-based spatial outliers: algorithms and applications (a summary of results). In *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 371–376. ACM.

[Shekhar et al., 2003] Shekhar, S., Lu, C.-T., and Zhang, P. (2003). A unified approach to detecting spatial outliers. *GeoInformatica*, 7(2):139–166.

[Shriparv, 2014] Shriparv, S. (2014). *Learning HBase*. Packt Publishing Ltd.

[Siegmund, 2013] Siegmund, D. (2013). *Sequential analysis: tests and confidence intervals*. Springer Science & Business Media.

[SLA, 2014] SLA, C. (2014). Cloud service level agreement standardisation guidelines. *European Commission, Brussels*.

[Smith, 2013] Smith, S. (2013). *Digital signal processing: a practical guide for engineers and scientists*. Newnes.

[Soule et al., 2005] Soule, A., Salamatian, K., and Taft, N. (2005). Combining filtering and statistical methods for anomaly detection. In *Proceedings of the 5th ACM SIGCOMM conference on Internet Measurement*, pages 31–31. USENIX Association.

[Strong et al., 1997] Strong, D. M., Lee, Y. W., and Wang, R. Y. (1997). *Data quality in context*. *Communications of the ACM*, pages 103–110.

[Sullivan and Heybey, 1998] Sullivan, M. and Heybey, A. (1998). A system for managing large databases of network traffic. In *Proceedings of USENIX*.

[Szewczyk et al., 2004] Szewczyk, R., Mainwaring, A., Polastre, J., Anderson, J., and Culler, D. (2004). An analysis of a large scale habitat monitoring application. In *Proceedings of the 2nd international conference on Embedded networked sensor systems*, pages 214–226. ACM.

[Tan et al., 2005] Tan, Y. L., Sehgal, V., and Shahri, H. H. (2005). Sensoclean: Handling noisy and incomplete data in sensor networks using modeling. Technical report, Technical report, University of maryland.

[Tasnim et al., 2017] Tasnim, S., Pissinou, N., and Iyengar, S. (2017). A novel cleaning approach of environmental sensing data streams. In *Consumer Communications & Networking Conference (CCNC), 2017 14th IEEE Annual*, pages 632–633. IEEE.

[Tatbul et al., 2003] Tatbul, N., Çetintemel, U., Zdonik, S., Cherniack, M., and Stonebraker, M. (2003). -load shedding in a data stream manager. In *Proceedings 2003 VLDB Conference*, pages 309–320. Elsevier.

[Tobler, 1970] Tobler, W. R. (1970). A computer movie simulating urban growth in the detroit region. *Economic geography*, 46(sup1):234–240.

[Tsay et al., 2000] Tsay, R. S., Peña, D., and Pankratz, A. E. (2000). Outliers in multivariate time series. *Biometrika*, 87(4):789–804.

[Van Phuong et al., 2006] Van Phuong, T., Hung, L. X., Cho, S. J., Lee, Y.-K., and Lee, S. (2006). An anomaly detection algorithm for detecting attacks in wireless sensor networks. In *Proceedings of the 4th IEEE international conference on Intelligence and Security Informatics*, pages 735–736. Springer-Verlag.

[Vitter, 1985] Vitter, J. S. (1985). *Random sampling with a reservoir*. *ACM Transactions on Mathematical Software (TOMS)*, pages 37–57.

[Vogel et al., 2011] Vogel, P., Greiser, T., and Mattfeld, D. C. (2011). Understanding bike-sharing systems using data mining: Exploring activity patterns. *Procedia-Social and Behavioral Sciences*, 20:514–523.

[Wand and Wang, 1996] Wand, Y. and Wang, R. Y. (1996). *Anchoring data quality dimensions in ontological foundations*. *Communications of the ACM*, 39(11):86–95.

[Wang, 1998] Wang, R. Y. (1998). A product perspective on total data quality management. *Communications of the ACM*, 41(2):58–65.

[Wang and Strong, 1996] Wang, R. Y. and Strong, D. M. (1996). *Beyond accuracy: What data quality means to data consumers*. *Journal of management information systems*, pages 5–33.

[Willcocks et al., 2013] Willcocks, L., Venters, W., and Whitley, E. (2013). *Moving to the Cloud Corporation: How to face the challenges and harness the potential of cloud computing*. Springer.

[Wong et al., 2002] Wong, W.-K., Moore, A., Cooper, G., and Wagner, M. (2002). Rule-based anomaly pattern detection for detecting disease outbreaks. In *AAAI/IAAI*, pages 217–223.

[Xu et al., 2015] Xu, K., Wang, F., Jia, X., and Wang, H. (2015). The impact of sampling on big data analysis of social media: A case study on flu and ebola. In *Global Communications Conference (GLOBECOM), 2015 IEEE*, pages 1–6. IEEE.

[Yu et al., 2016] Yu, Q., Jibin, L., and Jiang, L. (2016). An improved arima-based traffic anomaly detection algorithm for wireless sensor networks. *International Journal of Distributed Sensor Networks*, 12(1):9653230.

[Zaslavsky et al., 2013] Zaslavsky, A., Perera, C., and Georgakopoulos, D. (2013). Sensing as a service and big data. *arXiv preprint arXiv:1301.0159*.

[Zhang et al., 2008] Zhang, C., Luo, L., Xu, W., and Ledwith, V. (2008). Use of local moran's i and gis to identify pollution hotspots of pb in urban soils of galway, ireland. *Science of the total environment*, 398(1):212–221.

[Zhang et al., 2016] Zhang, H., Liu, J., Zhou, W., and Zhang, S. (2016). Sampling method in traffic logs analyzing. In *Intelligent Human-Machine Systems and Cybernetics (IHMSC), 2016 8th International Conference on*, volume 1, pages 554–558. IEEE.

[Zheng et al., 2013] Zheng, Z., Zhu, J., and Lyu, M. R. (2013). Service-generated big data and big data-as-a-service: an overview. In *Big Data (BigData Congress), 2013 IEEE International Congress on*, pages 403–410. IEEE.

[Zhu and Shasha, 2002] Zhu, Y. and Shasha, D. (2002). Statstream: Statistical monitoring of thousands of data streams in real time** work supported in part by us nsf grants iis-9988345 and n2010: 0115586. In *VLDB'02: Proceedings of the 28th International Conference on Very Large Databases*, pages 358–369. Elsevier.

[Zhuang and Chen, 2006] Zhuang, Y. and Chen, L. (2006). *In-network Outlier Cleaning for Data Collection in Sensor Networks*. In *CleanDB*.

[Zwetsloot et al., 2014] Zwetsloot, I. M., Schoonhoven, M., and Does, R. J. (2014). A robust estimator for location in phase i based on an ewma chart. *Journal of Quality Technology*, 46(4):302–316.

[Zwetsloot et al., 2015] Zwetsloot, I. M., Schoonhoven, M., and Does, R. J. (2015). A robust phase i exponentially weighted moving average chart for dispersion. *Quality and Reliability Engineering International*, 31(6):989–999.