Statistical Models for Querying and Managing Time-Series Data

THÈSE N^{O} 5705 (2013)

PRÉSENTÉE LE 21 JUIN 2013
À LA FACULTÉ INFORMATIQUE ET COMMUNICATIONS
LABORATOIRE DE SYSTÈMES D'INFORMATION RÉPARTIS
PROGRAMME DOCTORAL EN INFORMATIQUE, COMMUNICATIONS ET INFORMATION

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

POUR L'OBTENTION DU GRADE DE DOCTEUR ÈS SCIENCES

PAR

Saket SATHE

acceptée sur proposition du jury:

Prof. M. Grossglauser, président du jury
Prof. K. Aberer, directeur de thèse
Prof. R. Cheng, rapporteur
Prof. C. Koch, rapporteur
Prof. G. Trajcevski, rapporteur

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE
Suisse
2013

To my dear parents,
Vandana Sathe and late Keshav Sathe.

Acknowledgements

The first person I would like to thank is my thesis supervisor, Prof. Karl Aberer. He did an excellent job in supporting me throughout my PhD and at the same time gave me the opportunity to concentrate on the topics I liked most. I learnt a lot from Karl and I consider myself very lucky that I did my PhD in his lab.

I am very thankful to the members of my thesis committee: Prof Christoph Koch, Prof. Reynold Cheng, Prof. Goce Trajcevski and Prof. Matthias Grossglauser for their important comments and discussions to improve my dissertation.

I wish to thank all my colleagues whom I collaborated with during the work on this thesis, especially Dipanjan Chakraborty, Hoyoung Jeung, Thanasis Papaioannou, Sebastian Cartier, and Gleb Skobeltsyn. I thank all my colleagues from LSIR - we have a great team. A special thanks goes to Chantal who helped me sort out so many not only administrative issues.

I thank all my friends from the doctoral school group and beyond, for their support and for all the great moments we spent together, including all our travels, sports, adventures and parties: Surender, Dipanjan, Rammohan, Mehdi, Michele, Prem, Devika, Marc, Tri, Alexendra, Nishanth, Abhishek, Raj, Shobha, Laura, Shrinivas, Dinkar, Satish, Sanket, Sayali and many many others.

I would like to especially thank Nicola Pozza, who not only did a French-Hindi tandem with me for the last 4 years, but also perfectly translated the abstract of this thesis. I owe all my knowledge of the French language, Hindi Grammar, and Swiss politics to him.

Finally, I would like to thank my parents for their love and support, and for the many sacrifices that they've had to make. And, of course, I must never ever forget my wife Aishwarya. Not only is she the most intelligent and beautiful person I know, but she is also an amazing mother; our son Devansh is 10 months old and is learning to stand on his own.

Abstract

In recent years we are experiencing a dramatic increase in the amount of available time-series data. Primary sources of time-series data are sensor networks, medical monitoring, financial applications, news feeds and social networking applications. Availability of large amount of time-series data calls for scalable data management techniques that enable efficient querying and analysis of such data in real-time and archival settings. Often the time-series data generated from sensors (environmental, RFID, GPS, etc.), are imprecise and uncertain in nature. Thus, it is necessary to characterize this uncertainty for producing clean answers. In this thesis we propose methods that address these important issues pertaining to time-series data. Particularly, this thesis is centered around the following three topics:

Computing Statistical Measures on Large Time-Series Datasets.
Computing statistical measures for large databases of time series is a fundamental primitive for querying and mining time-series data [31, 81, 97, 111, 132, 137]. This primitive is gaining importance with the increasing number and rapid growth of time-series databases. In Chapter 3, we introduce the AFFINITY framework for efficient computation of statistical measures by exploiting the concept of *affine relationships* [113, 114]. Affine relationships can be used to infer a large number of statistical measures for time series, from other related time series, instead of computing them directly; thus, reducing the overall computational cost significantly. Moreover, the AFFINITY framework proposes an unified approach for computing several statistical measures at once.

Creating Probabilistic Databases from Imprecise Data.
A large amount of time-series data produced in the real-world has an inherent element of uncertainty, arising due to the various sources of imprecision affecting its sources (like, sensor data, GPS trajectories, environmental monitoring data, etc.). The primary sources of imprecision in such data are: imprecise sensors, limited communication bandwidth, sensor failures, etc. Recently there has been an exponential rise in the number of such imprecise sensors, which has led to an explosion of imprecise data. Standard database techniques cannot be used to provide clean and consistent answers in such scenarios. Therefore, probabilistic databases that factor-in the inherent uncertainty and produce clean answers are required. An important assumption

while using probabilistic databases is that each data point has a probability distribution associated with it. This is not true in practice — the distributions are absent. As a solution to this fundamental limitation, in Chapter 4 we propose methods for inferring such probability distributions and using them for efficiently creating probabilistic databases [116].

Managing Participatory Sensing Data.
Community-driven participatory sensing is a rapidly evolving paradigm in mobile geo-sensor networks. Here, sensors of various sorts (e.g., multi-sensor units monitoring air quality, cell phones, thermal watches, thermometers in vehicles, etc.) are carried by the community (public vehicles, private vehicles, or individuals) during their daily activities, collecting various types of data about their surrounding. Data generated by these devices is in large quantity, and geographically and temporally skewed. Therefore, it is important that systems designed for managing such data should be aware of these unique data characteristics.

In Chapter 5, we propose the ConDense (Community-driven Sensing of the Environment) framework for managing and querying community-sensed data [5, 19, 115]. ConDense exploits spatial smoothness of environmental parameters (like, ambient pollution [5] or radiation [2]) to construct statistical models of the data. Since the number of constructed models is significantly smaller than the original data, we show that using our approach leads to dramatic increase in query processing efficiency [19, 115] and significantly reduces memory usage.

Keywords: *time-series data management, statistical query processing, adaptive clustering, community sensing, probabilistic databases, affine transformations, view generation, approximate caching.*

Résumé

La quantité de données sous forme de série temporelle a augmenté de manière spectaculaire ces dernières années. Les sources principales de ces données proviennent de réseaux de capteurs, du monitoring médical, des applications financières, de flux d'actualités et des réseaux sociaux. Afin que ces quantités importantes de données issues des séries temporelles soient disponibles, des techniques de gestion de données extensibles permettant un traîtement des requêtes efficace et une analyse de ces données en temps réel et en tant qu'archives sont nécessaires. Ces données, quand elles sont issues de capteurs (environnementaux, RFID, GPS, etc.), sont toutefois souvent imprécises et peu fiables. Il est par conséquent nécessaire de caractériser cette incertitude, afin de pouvoir fournir des réponses fiables. Dans cette thèse, nous proposons certaines méthodes permettant de traiter ces importants problèmes liés aux données des séries temporelles. Cette thèse se concentre en particulier sur les trois sujets suivants:

Calcul de mesures statistiques sur des séries temporelles à large échelle: Le calcul des mesures statistiques de données de séries temporelles à large échelle est un prérequis indispensable à la récolte et à l'examen de ces données [31, 81, 97, 111, 132, 137]. Ce prérequis gagne en importance au fur et à mesure qu'augmentent la quantité et la taille des bases de données. Le chapitre 3 présente l'architecture AFFINITY, nécessaire au calcul efficace des mesures statistiques et basé sur le concept de relations affines [113, 114]. Les relations affines peuvent être utilisées pour déduire un nombre important de mesures, à partir d'autres séries temporelles qui leur sont liées, plutôt qu'en les calculant des séries originales; réduisant ainsi de manière drastique le calcul numérique global. De plus, l'architecture AFFINITY propose une approche unifiée pour calculer en une seule fois plusieurs mesures statistiques.

Création de bases de données probabilistes à partir de données imprécises: Une grande quantité de données de séries temporelles produites dans le monde réel comporte une part inhérente d'incertitude, en raison des diverses causes d'imprécision affectant leurs sources (par ex., les capteurs, la géolocalisation, les observatoires environnementaux, etc.). Les principales sources d'imprécisions proviennent de capteurs imprécis, de bandes passantes limitées, de pannes des capteurs, etc. Récemment, le nombre de capteurs imprécis a augmenté de manière exponentielle, ce qui a entraîné une explosion des données imprécises. Par ailleurs, les techniques habituelles utilisées pour les bases de données ne permettent pas de fournir des réponses fiables et cohérentes dans de tels cas de figure. Il est par conséquent nécessaire d'utiliser des bases de données probabilistes qui tiennent compte de cette incertitude et fournissent des réponses fiables. Quand ce genre de données est utilisé, il est souvent admis que chaque donnée est associée à une loi de probabilité. Ce n'est cependant pas le cas en pratique: ces lois de probabilité sont absentes. Comme solution à cette limitation fondamentale, nous proposons dans le chapitre 4 des méthodes permettant de déduire ces lois et de créer efficacement des bases de données utilisant les lois déduites [116].

Gestion de données de type *participatory sensing*: La méthode de *participatory sensing* est un modèle évoluant rapidement parmi les réseaux de géo-capteurs mobiles. Dans ce cas de figure, différentes sortes de capteurs (par exemple, des unités de capteurs multiples observant la qualité de l'air, des téléphones mobiles, des montres thermiques, des thermomètres installés dans les véhicules, etc.) sont transportés par les gens (véhicules publics et privés, individus) pendant leurs activités quotidiennes, recueillant divers types de données sur leur environnement. Les aspects spatio-temporelles des nombreuses informations collectées par ces capteurs sont fréquemment biaisées. C'est pourquoi il est important que les systèmes conçus pour gérer ces données tiennent compte de leurs caractéristiques propres.

Au chapitre 5, nous proposons l'architecture ConDense (*Community-driven Sensing of the Environment*) pour gérer et traîter des requêtes sur ce type de données [5, 19, 115]. ConDense exploite la régularité spatiale des paramètres environnementaux (par exemple, la pollution ambiante, le rayonnement, etc.), afin de construire les modèles statistiques de ces données. Comme le nombre de modèles construits est beaucoup plus faible que celui des données, nous pouvons montrer que l'utilisation de cette approche entraîne une augmentation spectaculaire de l'efficacité du traitement des requêtes [19, 115] et une diminution considérable de l'utilisation de la mémoire.

Mots-clés: gestion des données de séries temporelles, traitement de requête statistique, clustering adaptif, community sensing, base de données probabiliste, transformation affine, génération de vue, caching approximatif.

Contents

Dedication i

Acknowledgment i

Abstract i

Résumé i

Contents iii

List of Figures ix

List of Tables xiii

List of Algorithms xv

1 Introduction 1

1.1 The Time-Series Database Model 5

1.2 Contributions 6

1.3 Thesis Organization 8

1.4 Selected Publications 8

2 State of the Art 11

2.1 Introduction 11

2.1.1 Chapter Organization 13

2.2 Model-Based Data Acquisition 13

2.2.1 The Sensor Data Acquisition Query 14
2.2.2 Pull-Based Data Acquisition 14
2.2.2.1 In-Network Data Acquisition 14
2.2.2.2 Multi-Dimensional Gaussian Distributions 16
2.2.3 Push-Based Data Acquisition 17
2.2.3.1 PRESTO 17
2.2.3.2 Ken 19
2.2.3.3 A Generic Push-Based Approach 20
2.3 Model-Based Data Cleaning 20
2.3.1 Overview of the Data Cleaning System 21
2.3.2 Models for Data Cleaning 22
2.3.2.1 Regression Models 22
2.3.2.2 Probabilistic Models 24
2.3.2.3 Outlier Detection Models 25
2.3.3 Declarative Data Cleaning Approaches 25
2.4 Model-Based Query Processing 26
2.4.1 In-Network Query Processing 26
2.4.2 Model-Based Views 27
2.4.3 Symbolic Query Evaluation 28
2.4.4 Processing Queries over Uncertain Data 29
2.4.4.1 Dynamic Probabilistic Models 29
2.4.4.2 Static Probabilistic Models 30
2.4.5 Query Processing over Semantic States 31
2.4.6 Processing Event Queries 31
2.5 Model-Based Data Compression 32
2.5.1 Overview of Data Compression System 32
2.5.2 Methods for Data Segmentation 34
2.5.3 Piecewise Approximation 34
2.5.3.1 Swing and Slide Filters 35
2.5.3.2 Piecewise Linear Approximation 36
2.5.4 Compressing Correlated Data Streams 37
2.5.5 Multi-Model Data Compression 38

2.5.6 Orthogonal Transformations 38
2.5.6.1 Discrete Fourier Transform (DFT) 39
2.5.7 Lossless vs. Lossy Compression 40
2.6 Summary 40

3 Affinity: Efficiently Querying Statistical Measures on Time-Series Data 43
3.1 Introduction 43
3.1.1 Chapter Organization 46
3.2 Foundation 47
3.2.1 Statistical Measures 47
3.2.2 Query Types 49
3.2.3 Affine Transformations 49
3.3 Affine Clustering 51
3.3.1 Computing the Dot Product 51
3.3.2 Computing Other Measures 52
3.3.3 The AFCLST Clustering Algorithm 53
3.4 Computing Affine Relationships 56
3.4.1 Measure Computation Query 58
3.5 Indexing Affine Relationships 59
3.5.1 Scalar Projection (SCAPE) Index 60
3.5.2 Processing Threshold and Range Queries 62
3.5.3 Index-based Pruning for D-Measures 62
3.6 Experimental Evaluation 64
3.6.1 Analyzing Trade-Off 65
3.6.2 Impact of Online Environments 67
3.6.3 Scalability of the SYMEX Algorithm 68
3.6.4 Impact of using the SCAPE Index 68
3.7 Related Work 70
3.8 Conclusion 71

4 Creating Probabilistic Databases from Imprecise Time-Series Data 73
4.1 Introduction 73
4.1.1 Chapter Organization 76

4.2 Foundation . . . 76
4.2.1 Framework Overview . . . 76
4.2.2 Evaluation of Dynamic Density Metrics . . . 78
4.3 Naive Dynamic Density Metrics . . . 79
4.4 GARCH Metric . . . 80
4.4.1 The GARCH Model . . . 81
4.5 Enhanced GARCH Metric . . . 83
4.5.1 C-GARCH Model . . . 83
4.5.2 Successive Variance Reduction Filter . . . 84
4.6 Probabilistic View Generation . . . 86
4.6.1 σ–cache . . . 87
4.6.2 Constraint-Aware Caching . . . 88
4.7 Experimental Evaluation . . . 91
4.7.1 Comparison of Dynamic Density Metrics . . . 91
4.7.2 Impact of C-GARCH . . . 93
4.7.3 Impact of using σ–cache . . . 94
4.7.4 Verifying Time-Varying Volatility . . . 95
4.8 Related Work . . . 96
4.9 Conclusion . . . 97
Appendix 4.A Probabilistic Query Evaluation . . . 97

5 ConDense: Managing Data in Community-Driven Mobile Geosensor Networks 101
5.1 Introduction . . . 101
5.1.1 Chapter Organization . . . 102
5.2 Sensors, Deployment, and Data Collection . . . 103
5.3 Related Work . . . 104
5.4 Problem Characterization . . . 105
5.5 Non-Adaptive Methods for Model Cover Estimation . . . 108
5.5.1 Grid-Based Model Cover . . . 108
5.5.2 Kriging-Based Model Cover . . . 109
5.6 Adaptive Methods for Model Cover Estimation . . . 109
5.6.1 Adaptive DBSCAN . . . 110

5.6.2 Adaptive K-Means 112
5.6.3 Efficiently Maintaining the Model Cover 114
5.7 Experimental Evaluation 114
5.7.1 Error Analysis 115
5.7.2 Comparing Efficiency of Model Cover Estimation Methods 116
5.7.3 Analyzing Temporal Validity of Model Cover 118
5.8 Conclusion 118

6 Conclusion and Future Directions 121

6.1 Conclusion 121
6.2 Future Directions 122
6.2.1 AFFINITY 122
6.2.2 Creating Probabilistic Databases 123
6.2.3 ConDense 124

List of Symbols 125

Bibliography 129

Curriculum Vitae 139

List of Figures

1.1 A sensor data processing system. 1

1.2 Time-series database table containing the data values. The position of a data source w_j is denoted as (x_j, y_j). In cases where the data sources are assumed to be stationary, the position can also be stored using a foreign-key relationship between w_j and (x_j, y_j). But, for simplicity, we assume that the `sensor_data` table is in a denormalized form. 5

2.1 Various time-series data management tasks performed using models. (a) to improve data acquisition efficiency, a function is fitted to the first three values, and the remaining values (shown dotted) are not acquired from the data sources (e.g., sensors), since they are within a threshold δ, (b) data values are cleaned by identifying outliers after fitting a linear model, (c) a query requesting the value at time t' can be answered using interpolation, (d) only the first and the last time-series data value can be stored as compressed representation of the data values. 12

2.2 Push- and pull-based methods for sensor data acquisition. 15

2.3 Toy example of a Semantic Routing Tree (SRT) and Acquisitional Query Processing (ACQP) over a sensor network with five sensors. Dotted arrows indicate the direction of query response. A given sensor appends its identifier w_i and value s_{ij} to the partial result, which is available from its sub-tree. 16

2.4 Architecture of time-series data cleaning system. 21

2.5 Detected anomalies based on a degree-2 Chebyshev regression. 23

2.6 An example of data cleaning based on a probabilistic model. 24

2.7 An example of anomaly detection using a SQL statement. 25

2.8 Example of the `RegModel` view with three sensors. `RegModel` is incrementally updated as new sensor values are acquired. 28

2.9 Particle filtering stores p weighted data values for each time instance t_i. . 30

2.10 The database schema for multi-model materialization. 33

2.11 Poor Man's Compression - MidRange (PMC-MR). 35

3.1 Stock prices for symbols INTC, AMD and MSFT on 2^{nd} January 2003. . . 44

3.2 Architecture of the Affinity framework. 48

3.3 Illustration of an affine transformation. 50

3.4 (a) the 2-D hyperplane $\mathcal{H}$, and (b) directional view of the hyperplane $\mathcal{H}$. . 54

3.5 Procedure for generating the pivot pairs. 56

3.6 Toy example demonstrating Observation 3.1. 60

3.7 Example of the SCAPE index for indexing a C-measure and a D- measure. 61

3.8 Index-based pruning for processing MET and MER queries on D-measures. 63

3.9 Efficiency and accuracy tradeoff for *sensor-data*. Note the logarithmic scale for the speedup in (c). 65

3.10 Efficiency and accuracy tradeoff for *stock-data*. Note the logarithmic scale for the speedup in (c). 66

3.11 Absolute time comparison for *stock-data*. Note the logarithmic scale for the speedup in (c). 66

3.12 Comparing query processing efficiency. 67

3.13 Scalability of the SYMEX algorithm. (a) *sensor-data* and (b) *stock-data*. . 68

3.14 Scalability of the index construction on *sensor-data*. 69

3.15 Comparing efficiency of the SCAPE index for the MET query. 69

3.16 Comparing efficiency of the SCAPE index for the MER query. 70

4.1 An example of creating a tuple-level probabilistic database from time-dependent probability distributions. 74

4.2 Architecture of the framework. 77

4.3 Examples of naive dynamic density metrics. 79

4.4 Regions of changing volatility in (a) ambient temperature and (b) relative humidity. 81
4.5 (a) Behavior of the GARCH model when window $\boldsymbol{s}_{l-1}^{H}$ contains erroneous values. (b) Result of using the C-GARCH model. 84
4.6 Showing sample run of the Successive Variance Reduction Filter (Algorithm 4.2). 85
4.7 An example illustrating that o_u remains unchanged under mean shift operations when two Gaussian distributions have equal variance. 88
4.8 Structure of the σ–cache. 89
4.9 Comparing quality of the dynamic density metrics. 92
4.10 Comparing efficiency of the dynamic density metrics. Note the logarithmic scale on the y-axis. 92
4.11 Effect of model order on *campus-data*. 93
4.12 Comparing C-GARCH and GARCH. (a) Percentage of erroneous values successfully detected and (b) average time for processing a single value. . 94
4.13 (a) Impact of using the σ–cache on efficiency. (b) Scaling behavior of the σ–cache. Note the exponential scale on the x-axis. 94
4.14 Verifying time-varying volatility. 96

5.1 Community-driven mobile geosensor network infrastructure. 103
5.2 Architecture of the ConDense framework. 106
5.3 Weighting scheme for Ad-DBS. (a) shaded area shows an example of two overlapping regions, (b) shows the regions with the corresponding sensor values s, (c) and (d) present the weighting functions K_o and κ_o used for interpolation. 112
5.4 Ad-KMN iterations on toy data. (a) the centroids of regions R_1 and R_2 are computed, after which models M_1 and M_2 are estimated. (b) since error $u_1 > \tau_n$ and $u_2 > \tau_n$, we add two new clusters R_3 and R_4 using k-means clustering algorithm. 113
5.5 Comparing the decrease in percentage error as the number of regions increase. *Unweighted Ad-DBS* denotes Ad-DBS without the weighting scheme of Eq. (5.5). Note the different ranges on the y-axis. 115

5.6 Comparing the percentage normal error for Ad-KMN and GRIB over randomly chosen windows $\mathcal{B}_c$. Note the different ranges on the y-axis. . . 116
5.7 Comparing efficiency of (a) model cover estimation and (b) processing a point query (interpolation) on *opensense*. 117
5.8 Comparing the memory requirement of all the model cover estimation methods. 117
5.9 Comparing temporal validity of the model cover produced by (a) Ad-KMN and (b) GRIB on *opensense*. 118

List of Tables

1.1 Summary of notations. 6

3.1 Summary of notations. 47
3.2 Choices of $\boldsymbol{\alpha}_q$ and $\boldsymbol{\beta}_{qd}$. The third column refers to the affine relationship $(\mathbf{A}, \boldsymbol{b})$ between p_q and e_{qd}. 62
3.3 Summary of the datasets. 65
3.4 Query processing speedup computed when the query returns the maximum size of the result set $\mathcal{A}_T$ or $\mathcal{A}_R$. 70

4.1 Summary of Notations. 78
4.2 Summary of Datasets 91

5.1 Characteristics of sensors and pollutants. 103
5.2 Summary of the Datasets. 104

List of Algorithms

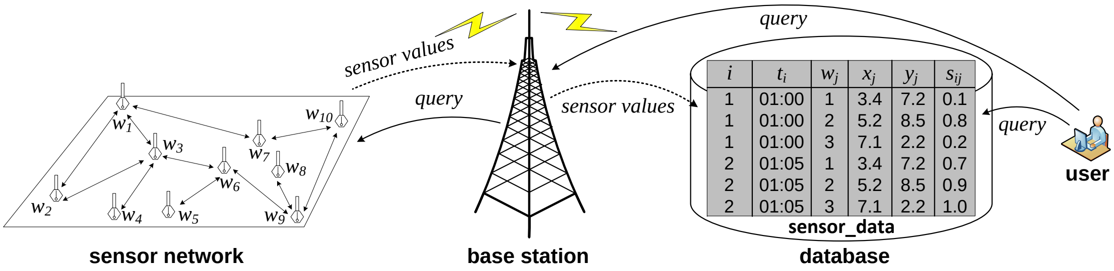
3.1 The AFCLST affine clustering algorithm. 55
3.2 The SYMEX algorithm. 58
4.1 Inferring $\hat{s}_l$ and $\hat{\sigma}_l^2$ using ARMA-GARCH. 83
4.2 The Successive Variance Reduction Filter. 86
5.1 The adaptive DBSCAN algorithm. 111
5.2 The adaptive k-means model cover method. 113

Chapter 1

Introduction

> *It is not the mountain we conquer, but ourselves.*
>
> Sir Edmund Hillary, 2003

Time-series data is becoming one of the fundamental primitives in many database operations. Therefore, there is an eminent need for scalable data management techniques that enable efficient querying and analysis of large amounts of time-series data in real-time and archival settings. Primary sources of time-series data are sensor networks, medical monitoring, financial applications, news feeds and social networking applications. Managing and querying such data poses several important challenges. Before diving into the details of these challenges, let us first understand the operation of one particular time-series data processing system, namely the sensor data processing system.



i	t_i	w_j	x_j	y_j	s_{ij}
1	01:00	1	3.4	7.2	0.1
1	01:00	2	5.2	8.5	0.8
1	01:00	3	7.1	2.2	0.2
2	01:05	1	3.4	7.2	0.7
2	01:05	2	5.2	8.5	0.9
2	01:05	3	7.1	2.2	1.0

Figure 1.1: A sensor data processing system.

Figure 1.1 shows an example of a sensor data processing system consisting of a network of sensors $w_1, w_2, \ldots, w_{10}$. Let us assume that these sensors are monitoring an environmental parameter, say, ambient temperature. These sensors transmit the sensed temperature values as time-series data to the base station. The base station transmits these values to a centralized database, shown in Figure 1.1 by the table called `sensor_data`. The user of this system can query the table `sensor_data` for obtaining answers or, if the queries are simplistic, he/she can directly query the sensor network.

During querying and managing time-series data, the sensor data processing system performs the following tasks:

- **Data Acquisition:** The sensors are remotely deployed and are often battery-powered. For this reason, the system has to acquire sensor values as efficiently as possible. In the existing literature several approaches have been proposed for this task [29, 38, 39, 82, 121]. Other proposals, like TinyDB [87, 88, 89], Cougar [134] and TiNA [117], advocate combining the task of data acquisition and query processing and collectively refer to it as *in-network query processing* or *acquisitional query processing.*
- **Data Cleaning:** Due to several reasons (poor weather conditions, faulty sensors, etc.), sensors often generate dirty or erroneous data. The sensor data processing system has to clean the sensor values by eliminating and interpolating the erroneous data values. This task is performed occasionally when the time-series data is streamed into the system. Several model-based time-series data cleaning techniques have been suggested in the literature [64, 91, 109, 116, 122].
- **Data Compression:** The data generated from sensor networks, like the one shown in Figure 1.1, is significantly large. At the same time, not all the data collected may be needed for processing user-queries [8, 29, 48, 111]. Therefore, the system has to optimally compress the collected data. This task is performed on the database side, by utilizing a wide-variety of signal processing techniques (refer Section 2.5). Some techniques, however, perform this task indirectly. Here, sensor values are transmitted only if they change considerably, leading to a collection of step functions that can be compressed efficiently [82, 121].
- **Data Retrieval:** In certain cases the users are interested in efficiently retrieving the data from the sensor data processing system. If the data is compressed, it may have to be reconstructed – at least partially – and presented to the user [21, 125]. Approaches like MauveDB [40] advocate to maintain model-based views in the form of regression models, and subsequently reconstruct the data using models as required. Other approaches propose methods for retrieving the time-series data based on similarity queries [6, 69, 83, 107, 108]. These techniques mainly rely on signal processing methods like DFT (Discrete Fourier Transform) or DWT (Discrete Wavelet Transform).
- **Query Processing:** The sensor data processing system processes many other queries than the ones discussed above. They include aggregation queries [48, 87, 89, 97], event queries [110], continuous queries [94, 95, 96], probabilistic queries [24, 25, 66, 100, 116, 126], etc. All these queries are processed over the `sensor_data` table.

While performing the above tasks there are several important and relevant challenges that we discuss in the sequel:

- **Data Scale:** There has been a tremendous increase in the number of sensors that are sensing different aspects of our environment or day-to-day lives. As a consequence, the data available from these devices is growing at a tremendous rate. For example, since environmental monitoring sensors are capable of operating at a sampling rate of 30 samples/minute, a single sensor can collect approximately 43K samples in one day. Another example are the accelerometers used in smartphones. These accelerometers operate at a much higher sampling frequency as compared to the environmental sensors [133]. The data generated by these sensors is mainly time-series data. The growing scale of this data opens up many interesting query processing problems.

- **Data Uncertainty:** Time-series data generated especially by sensors (environmental sensors, RFID, GPS, etc.) is uncertain due to several reasons: (a) imprecise or inaccurate sensors, (b) intermittent sensor failures, (c) background noise influencing the sensor values, (d) loss of communication that produces erroneous sensor values, (e) other types of sensor failures, for example, snow accumulation on the sensor, etc.

  One of the most effective ways to deal with imprecise and uncertain data is to employ probabilistic approaches. In recent years there have been a plethora of methods for managing and querying uncertain data [24, 32, 34, 57, 100, 110, 124]. These methods are typically based on the assumption that probability distributions used for processing queries are available; however, this need not be always true – in many cases the distributions are absent.

- **Data Abstraction:** One important step for managing and querying sensor network data is to create abstractions of the data in the form of models. These models can then be stored, retrieved, and queried, as required. There has been significant amount of prior literature on using models for query processing [14, 24, 40, 53, 110]. These approaches do not consider the community sensing scenario, where the data is generated by sensors carried by the community (public or private vehicles, individuals). Admittedly, there has been a lack of understanding on developing reliable models, considering the unique characteristics (e.g., high spatio-temporal skewness) of community-sensed data.

In this thesis we present contributions addressing each of the three above mentioned challenges, which are applicable in different data processing stages. Below we briefly discuss our proposed solutions:

1. **Handling Large-Scale Data:** One important query that is sensitive to the scale of the data, involves computing statistical measures (for e.g., the correlation coefficient matrix) on large time-series datasets [31, 81, 97, 111, 132, 137]. The trivial solution to this problem is to compute the correlation coefficient for each pair of time series from scratch. But, since the number of pairs is quadratic, computing the pairwise correlation coefficients from scratch becomes infeasible for large time-series datasets. As a solution to this problem, in Chapter 3 we introduce a framework for efficient computation of statistical measures by exploiting the concept of *affine relationships*. Affine relationships can be used to infer statistical measures for time series from other related time series instead of directly computing them; thus, reducing the overall

computation cost significantly. The resulting methods show at least one order of magnitude improvement over the best known methods.

2. **Characterizing Data Uncertainty:** Creating probabilistic data that will capture the uncertainty is an every-challenging problem. Prior work on this problem has only limited scope for domain-specific applications, such as handling duplicated tuples [10, 56] and deriving structured data from unstructured data [55]. Evidently, a wide range of applications still lack the benefits of existing query processing techniques that require probabilistic data. Time-series data is one important example where probabilistic data processing is currently not widely applicable due to the lack of proper probabilistic description of the uncertain data values. One of the most important challenges in creating a probabilistic database from time series is to deal with evolving probability distributions, since time series often exhibit highly irregular dependencies on time [32, 126]. For example, temperature changes dramatically around sunrise and sunset, but changes only slightly during the night. This implies that the probability distributions that are used as the basis for deriving probabilistic databases also change over time, and thus must be computed dynamically.

   In order to capture the evolving probability distributions of time series, in Chapter 4 we introduce various *dynamic density metrics*, each of which dynamically inferring time-dependent probability distributions from a given time series. We identify and adopt a novel class of dynamical models from the time-series literature, known as the GARCH (Generalized AutoRegressive Conditional Heteroskedasticity) model [116]. We show that the GARCH model can be used for accurately inferring dynamic probability distributions. The distributions derived by these dynamic density metrics are used for creating probabilistic databases. Furthermore, for enhancing the efficiency of database creation, we propose a caching mechanism called σ-cache. We prove theoretical guarantees that are used for setting the cache parameters. Lastly, we show that by using the σ-cache the performance of creating probabilistic databases is enhanced by at least an order of magnitude as compared to the baseline approaches.

3. **Creating Succinct Abstractions:** Participatory sensing in Community-driven Mobile GeoSensor Networks (CGSN) is a rapidly evolving paradigm. Here, sensors of various sorts (e.g., multi-sensor units monitoring air quality, cell phones, thermal watches, etc.) are carried by the community (public or private vehicles, individuals) during their daily activities, collecting various types of data about their surrounding. Data generated by these devices is in large quantity and geographically and temporally skewed. For example, in highly populated areas or city centers we will have large number of data values and vice-versa or in the night the sampling frequency dramatically reduces due the sensors being turned off or stationary.

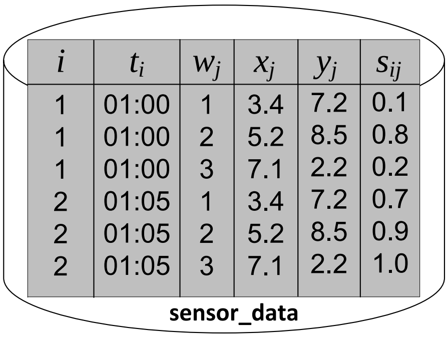
   Thus, creating succinct abstractions in the form of models from the data generated by CGSNs is a new and challenging problem. In an effort to address this challenge, in

Chapter 5 we propose various approaches for modeling the data from a community-driven mobile geo-sensor network. This data is typically collected over a large geographical area with mobile sensors having uncontrolled or semi-controlled mobility. We propose adaptive techniques that take into account such mobility patterns and produce an accurate representation of the sensed spatio-temporal phenomenon.

Next, we start by defining the time-series data model and the technical terms that are used in the rest of the thesis. We also describe the framework and establish a handful of basic notations.

1.1 The Time-Series Database Model

In this section we will give the time-series database model and the basic notation that will be consistently used in the rest of the thesis. In this thesis we consider a database consisting of time-series data. An example of such database is the `sensor_data` table shown in Figure 1.2. Each column in the `sensor_data` table is a function of time. Another example of a time-series database is a collection of stock quotes. In such a collection each stock's values are ordered according to time.



i	t_i	w_j	x_j	y_j	s_{ij}
1	01:00	1	3.4	7.2	0.1
1	01:00	2	5.2	8.5	0.8
1	01:00	3	7.1	2.2	0.2
2	01:05	1	3.4	7.2	0.7
2	01:05	2	5.2	8.5	0.9
2	01:05	3	7.1	2.2	1.0

sensor_data

Figure 1.2: Time-series database table containing the data values. The position of a data source w_j is denoted as (x_j, y_j). In cases where the data sources are assumed to be stationary, the position can also be stored using a foreign-key relationship between w_j and (x_j, y_j). But, for simplicity, we assume that the `sensor_data` table is in a denormalized form.

Now, let us consider the formal definition of time-series data and time-series data source.

DEFINITION 1.1: **Time-series data and time-series data source.** A time series or time-series data is a sequence of real numbers, where these numbers are ordered according to time of their occurrence, arrival or acquisition. A time-series data source[1] is defined as the entity that is producing time-series data.

For example, in a scenario where a sensor is monitoring ambient temperature, the temperature sensor is the time-series data source and the data generated by it is the time-series data. Now, we describe our model of a time-series database. The time-series

---

[1] In this thesis we use *time-series data source* and *data source* interchangeably.

database considered in this thesis consists of a set of time-series data sources denoted as $\mathcal{W} = \{w_j | 1 \leq j \leq n\}$. The value obtained from a source w_j at time t_i is denoted as s_{ij}, which is a real number. In addition, note that we use w_j, where $j = (1, \ldots, n)$, as data source identifiers. In most of the cases that we consider in this thesis, the sampling interval is considered uniform; $t_{i+1} - t_i$ is same for all the values of $i \geq 1$. In such cases, the time stamps t_i become irrelevant, and it is sufficient to use only the index i for denoting the time axis.

The random variable associated with the data value s_{ij} is denoted as R_{ij}. We denote the row vector of all the data values observed at time t_i as $\boldsymbol{r}_i^\top \in \mathbb{R}^n$; the vector-valued random variable associated with $\boldsymbol{r}_i$ is denoted as $\boldsymbol{R}_i$. In some instances (refer Chapter 3) we consider n time series, each of fixed length m. In these cases, we can compose a matrix consisting of m rows by concatenating the n column vectors $\boldsymbol{s}_1, \boldsymbol{s}_2, \ldots, \boldsymbol{s}_n$ as $\mathbf{S} = [\boldsymbol{s}_1, \boldsymbol{s}_2, \ldots, \boldsymbol{s}_n] \in \mathbb{R}^{m \times n}$. We refer to this matrix $\mathbf{S}$ as the *data matrix.*

Table 1.1: Summary of notations.

Symbol	**Description**
$\mathcal{W}$	Time-series database consisting of sources w_j, where $j = (1, \ldots, n)$.
w_j	Time-series data source in the database $\mathcal{W}$.
s_{ij}	Data value observed by the source w_j at time t_i, such that $s_{ij} \in \mathbb{R}$.
$\boldsymbol{s}_j$	Column vector of all the data values observed by the data source w_j, such that $s_j \in \mathbb{R}^m$.
$\mathbf{S}$	Data matrix $\mathbf{S} = [s_1, s_2, \ldots, s_n] \in \mathbb{R}^{m \times n}$.
$\boldsymbol{r}_i$	Row vector of all the data values observed at time t_i, such that $\boldsymbol{r}_i^\top \in \mathbb{R}^n$.
R_{ij}	Random variable associated with the data value s_{ij}.
$\boldsymbol{R}_i$	Random variable associated with the row vector $\boldsymbol{r}_i$.

We summarize the basic notation used in this thesis in Table 1.1. Naturally, these are not the only notations that are required for the thesis. A complete list of symbols used in this thesis can be found from page 125 onward. Moreover, for the sake of convenience, in each chapter we summarize the notation used in it, by including a table like Table 1.1.

1.2 Contributions

Our proposed solutions lead to the following concrete contributions:

- **End-to-end solution for creating probabilistic databases:** The first contribution of this thesis is a generic end-to-end solution for creating probabilistic databases from arbitrary imprecise time-series data. Specifically, we introduce various dynamic density metrics for associating tuples of raw time-series data values with probability distributions. Then, we define a qualitative measure known as the *density distance* that quantifies the effectiveness of the dynamic density metrics. This serves as an important measure for indicating the quality of probabilistic databases derived using a dynamic density metric. Since time-series data sources often deliver error prone data values we propose effective enhancements that make the dynamic density metrics robust against unclean data.

We then present approaches that allow applications to efficiently create probabilistic databases by using a SQL-like syntax. We present a generic framework comprising of a malleable query provisioning layer, called Ω–View builder, which allows us to create probabilistic databases with minimal effort. Furthermore, we propose a space- and time-efficient caching mechanism, called σ–cache, which produces manyfold improvement in performance. In addition, we prove useful guarantees for effectively setting the cache parameters.

- **Methods for fast computation and querying of statistical measures:** To the best of our knowledge, the Affinity framework proposed in Chapter 3 is the first work that exploits multi-dimensional affine transformations for time-series data management. The fundamental contribution here is the introduction of affine relationships for efficiently querying and computing several statistical measures. Compared to the existing state of the art methods [97, 137], which use the Discrete Fourier Transform (DFT) to approximate the correlation coefficient, our methods use affine relationships that are amenable to indexing, thus resulting in orders of magnitude performance improvement over the state of the art methods. Furthermore, our methods are more general and can be used for computing many other statistical measures with even better performance gains as compared to the correlation coefficient.

  The technical contributions that we make on this topic are as follows:

  - We propose a distance metric called the *least significant frobenius distance* (LSFD) for characterizing the quality of affine relationships.
  - We present a novel clustering algorithm called *affine clustering* (AFCLST) that is capable of clustering the given data, such that high-quality (low LSFD) affine relationships could be found within the cluster members.
  - We introduce an efficient algorithm called the *systematic exploration algorithm* (SYMEX), that generates high-quality affine relationships on-the-fly, by utilizing the output of the AFCLST clustering algorithm.
  - We show that indexing affine relationships with the SCAPE index that we introduced in [114], results in orders of magnitude performance improvement for processing statistical queries.

- **Adaptive techniques for managing data in CGSNs:** Our main contribution on this topic are the adaptive strategies for modeling data from a CGSN. Our techniques discover spatial areas that can be modeled using a single or multiple statistical models. To capture the phenomena with high fidelity, our strategies adapt to the changing nature of the sensed phenomena by adjusting the geographical granularity of the models. In addition, our strategies have user-defined approximation error thresholds, which can be used for adjusting the level of geographical granularity and quality of the models produced by our approaches. On the temporal dimension, we use slack functions (on the models) to time-out low quality models (i.e., models that no longer fit the current data).

- **Extensive experimental evaluation on real datasets:** One of the most notable contributions of this thesis is the extensive experimental evaluation of all the proposed approaches on real datasets. To increase empirical confidence in our methods, the datasets that we use are obtained from various data sources: (a) stock markets, (b) environmental monitoring applications, (c) sensor networks and (d) GPS trajectories.

1.3 Thesis Organization

We begin by surveying the state of the art techniques that are relevant to this thesis in Chapter 2. This is followed by a discussion of the AFFINITY framework for fast computation of various statistical measures in Chapter 3. Then we present the SCAPE index structure, which is used for processing threshold and range queries for a large number of statistical measures in Section 3.5. Statistical time-series models for efficiently creating probabilistic databases are presented in Chapter 4. Tunable caching techniques for efficiently generating probabilistic views are presented in Section 4.6, along with provable accuracy and memory guarantees. The issue of modeling and querying community-sensed data over large geographical areas is covered in Chapter 5. Chapter 5 also introduces the ConDense framework designed for processing such data. Lastly, we summarize and conclude this thesis in Chapter 6, followed by detailed suggestions for future work.

1.4 Selected Publications

During the course of this thesis several research papers were published, but, this thesis is mainly based on the following research papers:

- S. Sathe, T. Papaioannou, H. Jeung, and K. Aberer. *Managing and Mining Sensor Data.* Springer, 2012, ch. A survey of model-based sensor data acquisition and management, ed. Charu Aggarwal. (Chapter 2)

- S. Sathe and K. Aberer. AFFINITY: Efficiently querying statistical measures on time-series data. In *ICDE* (to appear), 2013. (Chapter 3)

- S. Sathe, H. Jeung, and K. Aberer. Creating probabilistic databases from imprecise time-series data. In *ICDE*, pages 327–338, 2011. (Chapter 4)

- S. Sathe, S. Cartier, D. Chakraborty, and K. Aberer. Effectively Modeling Data from Large-area Community Sensor Networks. In *IPSN*, pages 95–96, 2012. (Chapter 5)
- S. Cartier, S. Sathe, D. Chakraborty, and K. Aberer. ConDense: Managing data in community-driven mobile geosensor networks. In *IEEE SECON*, 2012. (Chapter 5)

Chapter 2

State of the Art

> *All models are wrong. But some are useful.*
>
> George Box, 1979

2.1 Introduction

In this chapter, we survey a large number of state-of-the-art model-based techniques for querying and mining time-series data. These techniques use mathematical models for solving problems pertaining to time-series data management. Model-based techniques use different types of models: statistical, signal processing, regression-based, machine learning, probabilistic, or time series. These models serve various purposes in time-series data management.

We review four broad categories of time-series data management tasks: data acquisition, data cleaning, query processing, and data compression. These tasks are pictorially summarized in the toy example shown in Figure 2.1. From Figure 2.1, it is interesting to note how a single type of statistical model (linear regression) can be used for performing these various tasks. For each task considered in this chapter, we extensively discuss various, well-researched model-based solutions. Following is the detailed discussion on the time-series data management tasks covered in this chapter:

- **Data Acquisition:** This task is mainly relevant for data that is acquired or collected from remotely located sources. In wireless sensor networks, the primary objective of the data acquisition task is to attain energy efficiency. This objective is driven by the fact that most remotely located data sources are battery-powered and are located in inaccessible locations (e.g., environmental monitoring sensors are sometimes located at high altitudes and are surrounded by highly inaccessible terrains).

  In the literature, there are two major types of acquisition approaches: pull-based and push-based. In the pull-based approach, data is only acquired at a user-defined

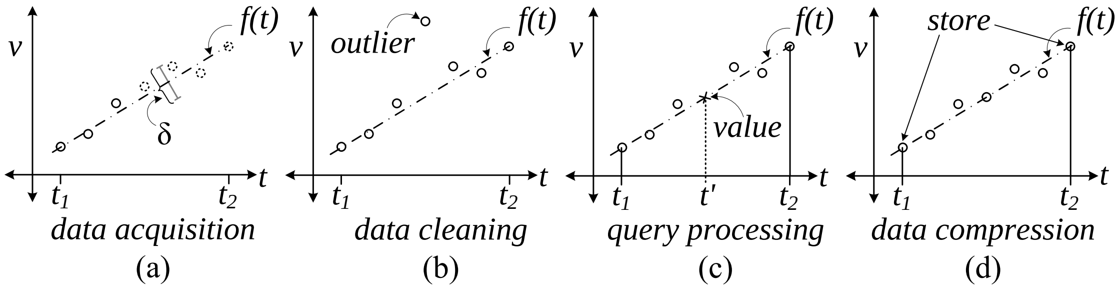


Figure 2.1: Various time-series data management tasks performed using models. (a) to improve data acquisition efficiency, a function is fitted to the first three values, and the remaining values (shown dotted) are not acquired from the data sources (e.g., sensors), since they are within a threshold δ, (b) data values are cleaned by identifying outliers after fitting a linear model, (c) a query requesting the value at time t' can be answered using interpolation, (d) only the first and the last time-series data value can be stored as compressed representation of the data values.

frequency of acquisition. On the other hand, in the push-based approach, the data sources and the base station agree on an expected behavior; data sources only send data to the base station if the data values deviate from such expected behavior. In this chapter, we cover a representative collection of model-based data acquisition approaches [9, 29, 38, 39, 40, 53, 54, 82, 127].

- **Data Cleaning:** As discussed in Chapter 1, the data obtained from the data acquisition task is often erroneous. Model-based approaches for data cleaning often use a model to infer the most probable data value. Then the raw data value is marked erroneous or outlier if it deviates significantly from the inferred data value. Another important approach for data cleaning is known as declarative data cleaning [64, 91, 109]. In this approach, the user registers SQL-like queries that define constraints over the data values. Data values are marked as outliers when these constraints are violated. In addition to these methods, we also discuss many other data cleaning approaches [44, 47, 63, 105, 126, 138]
- **Query Processing:** Obtaining desired answers by processing queries is another important aspect in time-series data management. In this chapter, we discuss the most significant model-based techniques for query processing. One of the objectives of these techniques is to process queries by accessing/generating minimal amount of data [14, 125]. Model-based methods that access/generate minimal data, and also handle missing values in data, use models for creating an abstraction layer over the time-series database [40, 66]. Other approaches model the data values by a hidden Markov model (HMM), associating state variables to the data values. It is more efficient to process queries over the state variables, which are less in number as compared to the data values [14]. Furthermore, there are approaches that use

dynamic probabilistic models (DPMs) for modeling spatio-temporal evolution of the data [60, 66], and the estimated DPMs are used for query processing.

- **Data Compression:** Eliminating redundancy by compressing time-series data for various purposes (like, storage, query processing, etc.) becomes one of the most challenging tasks. Model-based time-series data compression proposes a large number of techniques, mainly from the signal processing literature, for this task [6, 22, 46, 107, 137]. Many approaches assume that the user provides an accuracy bound, and based on this bound the sensor data is approximated, resulting in compressed representations of the data [48]. A large number of other techniques exploit the fact that time-series data is often correlated; thus, this correlation can be used for approximating one data stream with another [12, 48, 102, 129].

One of the main source of time-series data considered in this chapter is the data generated by sensors that are sensing physical attributes. It is well-known that many physical attributes, such as, ambient temperature or relative humidity, vary smoothly. As a result of this smoothness, time-series data generated from sensors sensing such physical attributes typically exhibits the following properties: (a) it is continuous (although we only have a finite number of samples), (b) it has finite energy or it is band-limited, (c) it exhibits Markovian behavior or the value at a time instant depends only on the value at a previous time instant. Most model-based techniques exploit these properties for efficiently performing various tasks.

2.1.1 Chapter Organization

The rest of the chapter is organized as follows. In Section 2.2, we discuss important techniques for time-series data acquisition. In Section 2.3, we survey model-based time-series data cleaning techniques, both on-line and archival. Model-based query processing techniques are discussed in Section 2.4. In Section 2.5, model-based compression techniques are surveyed. At the end, Section 2.6 contains a summary of the chapter along with conclusions.

2.2 Model-Based Data Acquisition

In this section, we discuss various techniques for model-based[1] data acquisition. The main sources used for acquiring data for the techniques described in this section are battery-powered sensors. We only consider sensors as the data sources, ignoring issues like connectivity, sleep scheduling, etc. Additionally, since many approaches proposed in Chapter 3, Chapter 4, and Chapter 5 use sensor data, here we review the techniques for efficiently acquiring sensor data. Particularly, we discuss pull- and push-based sensor data acquisition methods. In general, model-based sensor data acquisition techniques are designed for tackling the following challenges:

---

[1] We use *model-based* and *model-driven* interchangeably.

- **Energy Consumption:** Transmitting values from a sensor requires high amount of energy. In contrast, since most sensors are battery-powered, they have limited energy resources. Thus, a challenging task is to minimize the number of samples transmitted from the sensors. Hence one use of models for selecting sensors, such that user queries can be answered with reasonable accuracy using the data acquired from the selected sensors [9, 38, 39, 53, 54].
- **Communication Cost:** Another energy-intensive task is to communicate the sensed values to the base station. There are, therefore, several model-based techniques proposed in the literature for reducing the communication cost, and maintaining the accuracy of the sensed values [29, 40, 82, 127].

2.2.1 The Sensor Data Acquisition Query

Let us consider the time-series database model described in Section 1.1 (page 5). Consider the same scenario where the sensors w_j are monitoring the ambient temperature and the sensed data values are stored in the `sensor_data` table.

Sensor data acquisition can be defined as the processes of creating and continuously maintaining the `sensor_data` table. In existing literature many techniques have been proposed for creating and maintaining the `sensor_data` table [29, 38, 39, 82, 87, 88, 89, 117, 134]. We shall discuss these techniques briefly, describing their important characteristics and differences with other techniques. We use the sensor data acquisition query shown in Query 2.1 for discussing how different sensor data acquisition approaches process such a query. Query 2.1 triggers the acquisition of ten sensor values s_{ij} from the sensor w_j at a sampling interval of one second, and it is a typical sensor data acquisition query used by many methods for creating and maintaining the `sensor_data` table.

`SELECT` w_j, s_{ij} `FROM sensor_data SAMPLE INTERVAL 1s FOR 10s`

Query 2.1: Sensor data acquisition query.

2.2.2 Pull-Based Data Acquisition

Recall there are two major approaches for data acquisition: pull-based and push-based (refer Figure 2.2). In the pull-based approach, the user defines the interval and frequency of data acquisition. Pull-based systems only follow the user's requirements, and pull sensor values as defined by the queries. For example, using the `SAMPLE INTERVAL` clause of Query 2.1, users can specify the number of samples and the frequency at which the samples should be acquired.

2.2.2.1 In-Network Data Acquisition

This approach of sensor data acquisition is proposed by TinyDB [87, 88, 89], Cougar [134] and TiNA [117]. They tightly link query processing and sensor data acquisition. Due to the lack of space, we shall only discuss TinyDB in this subsection.

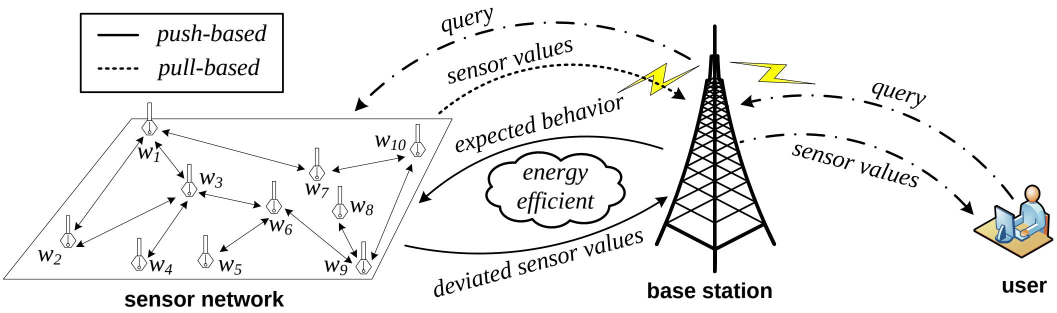


Figure 2.2: Push- and pull-based methods for sensor data acquisition.

TinyDB refers to its in-network query processing paradigm as *Acquisitional Query Processing* (ACQP). Let us start by discussing how ACQP processes Query 2.1. The result of Query 2.1 is similar to the table shown in Figure 1.2. The only difference, as compared to Figure 1.2, is that the result of Query 2.1 contains $10 \times n$ rows. The naïve method of executing Query 2.1 is to simultaneously poll each sensor for its value at the sampling interval and for the duration specified by the query. This method may not work due to limited range of radio communication between individual sensors and the base station.

Data Acquisition using Semantic Overlays: TinyDB proposes a tree-based overlay that is constructed using the sensors $\mathcal{W}$. This tree-based overlay is used for aggregating the query results from the leaf nodes to the root node. The overlay network is especially built for efficient data acquisition and query processing. TinyDB refers to its tree-based overlay network as *Semantic Routing Trees* (SRTs). A SRT is constructed by flooding the sensor network with the *SRT build request*. This request includes the attribute (ambient temperature), over which the SRT should be constructed. Each sensor w_j, which receives the build request, has several choices for choosing its parent: (a) if w_j has no children, which is equivalent to saying that no other sensor has chosen w_j as its parent, then w_j chooses any sensor as its parent and sends its current value s_{ij} to the chosen parent in a *parent selection message*, or (b) if w_j has children, it sends a parent selection message to its parent indicating the range of ambient temperature values that its children are covering. In addition, it locally stores the ambient temperature values from its children along with their sensor identifiers.

Next, when Query 2.1 is presented to the root node of the SRT, it forwards the query to its children and prepares for receiving the results. At the same time, the root node also starts processing the query locally (refer Figure 2.3). The same procedure is followed by all the intermediate sensors in the SRT. A sensor that does not have any children, processes the query and forwards the value of s_{ij} to its parent. All the collected sensor values s_{ij} are finally forwarded to the root node, and then to the user. This completes the processing of the sensor data acquisition query (Query 2.1). The SRT, moreover, can also be used for optimally processing aggregation, threshold, and event based queries. We shall return to this point later in Section 2.4.1.

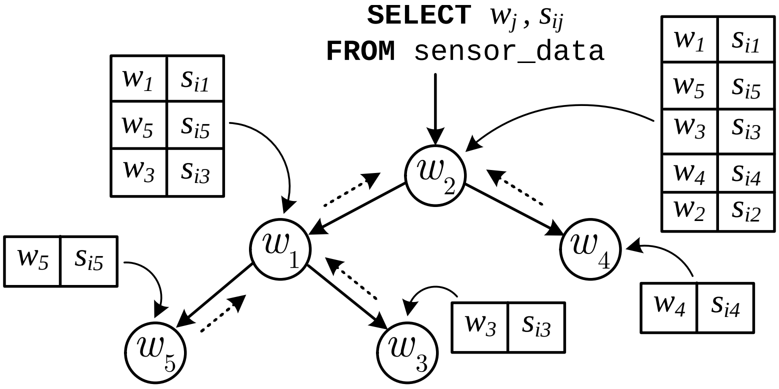


Figure 2.3: Toy example of a Semantic Routing Tree (SRT) and Acquisitional Query Processing (ACQP) over a sensor network with five sensors. Dotted arrows indicate the direction of query response. A given sensor appends its identifier w_i and value s_{ij} to the partial result, which is available from its sub-tree.

2.2.2.2 Multi-Dimensional Gaussian Distributions

The Barbie-Q (BBQ) system [38, 39] employs multi-variate Gaussian distributions for sensor data acquisition. BBQ maintains a multi-dimensional Gaussian probability distribution over all the sensors in $\mathcal{W}$. Data is acquired only as much as it is required to maintain such a distribution. Sensor data acquisition queries specify certain confidence that they require in the acquired data. If the confidence requirement cannot be satisfied, then more data is acquired from the sensors, and the Gaussian distribution is updated to satisfy the confidence requirements. The BBQ system models the sensor values using a multi-variate Gaussian probability density function (pdf) denoted as $\mathbb{P}(R_{i1}, R_{i2}, \ldots, R_{in})$, where $R_{i1}, R_{i2}, \ldots, R_{in}$ are the random variables associated with the sensor values $s_{i1}, s_{i2}, \ldots, s_{in}$ respectively. This *pdf* assigns a probability for each possible assignment of the sensor values s_{ij}. Now, let us discuss how the BBQ system processes Query 2.1.

In BBQ, the inferred sensor value of sensor w_j, at each time t_i, is defined as the mean value of R_{ij}, denoted $\bar{s}_{ij}$. For example, at time t_1, the inferred sensor values of the ambient temperature are $\bar{s}_{11}, \bar{s}_{12}, \ldots, \bar{s}_{1n}$. The BBQ system assumes that queries, like Query 2.1, provide two additional constraints: (i) error bound ϵ, for the values $\bar{s}_{ij}$, and (ii) the confidence $1 - \delta$ with which the error bound should be satisfied. Admittedly, these additional constraints are for controlling the quality of the query response.

Suppose we already have a pdf before the first time instance t_1, then the confidence of the sensor value s_{1j} is defined as the probability of the random variable R_{1j} lying in between $\bar{s}_{1j} - \epsilon$ and $\bar{s}_{1j} + \epsilon$, and is denoted as $\mathrm{P}(R_{1j} \in [\bar{s}_{1j} - \epsilon, \bar{s}_{1j} + \epsilon])$. If the confidence is greater than $1 - \delta$, then we can provide a probably approximately correct value for the temperature, without spending energy in obtaining a sample from sensor w_j. If a sensor's confidence is less than $1 - \delta$, then we should obtain one or more samples from the sensor (or other correlated sensors), such that the confidence bound is satisfied. In fact, it is clear that there could be potentially many sensors for which the confidence bound may not hold.

As a solution to this problem, the BBQ system proposes a procedure to chose the sensors for obtaining sensor values such that the confidence bound specified by the query is satisfied. First, the BBQ system samples from all the sensors $\mathcal{W}$ at time t_1, then it computes the confidence $\mathsf{B}_j(\mathcal{W})$ that it has in a sensor w_j as follows:

$$\mathsf{B}_j(\mathcal{W}) = \mathrm{P}(R_{1j} \in [\bar{s}_{1j} - \epsilon, \bar{s}_{1j} + \epsilon] | \boldsymbol{r}_1), \tag{2.1}$$

where $\boldsymbol{r}_1$ is the row vector of all the sensor values at time t_1. Second, for choosing sensors to sample, the BBQ system poses an optimization problem of the following form:

$$\min_{\mathcal{W}_o \subseteq \mathcal{W} \text{ and } \mathsf{B}(\mathcal{W}_o) \geq 1-\delta.} \mathcal{C}(\mathcal{W}_o), \tag{2.2}$$

where $\mathcal{W}_o$ is the subset of sensors that will be chosen for sampling, $\mathcal{C}(\mathcal{W}_o)$ and $\mathsf{B}(\mathcal{W}_o) = \frac{1}{|\mathcal{W}_o|} \sum_{j:w_j \in \mathcal{W}_o} \mathsf{B}_j(\mathcal{W})$ are respectively the total cost (or energy required) and average confidence for sampling sensors $\mathcal{W}_o$. Since the problem in Eq. (2.2) is NP-hard, BBQ proposes a greedy solution to solve this problem. Details of this greedy algorithm can be found in [39]. By executing the proposed greedy algorithm, BBQ selects the sensors for sampling, then it updates the Gaussian distribution, and returns the mean values $\bar{s}_{11}, \bar{s}_{12}, \ldots, \bar{s}_{1m}$. These mean values represent the inferred values of the sensors at time t_1. This operation when performed ten times at an interval of one second generates the result of the sensor data acquisition query (Query 2.1).

2.2.3 Push-Based Data Acquisition

Both TinyDB and BBQ are pull-based in nature: in these systems the central server/base station decides when to acquire sensor values from the sensors. On the other hand, in push-based approaches, the sensors autonomously decide when to communicate sensor values to the base station (refer Figure 2.2). Here, the base station and the sensors agree on an expected behavior of the sensor values, which is expressed as a model. If the sensor values deviate from their expected behavior, then the sensors communicate only the deviated values to the base station.

2.2.3.1 PRESTO

The Predictive Storage (PRESTO) [82] system is an example of the push-based data acquisition approach. One of the main arguments that PRESTO makes against pull-based approaches is that such approaches will be unable to observe any unusual or interesting patterns between any two pull requests. Moreover, increasing the pull frequency for better detection of such patterns, increases the overall energy consumption of the system.

The PRESTO system contains two main components: PRESTO proxies and PRESTO sensors. As compared to the PRESTO sensors, the PRESTO proxies have higher computational capability and storage resources. The task of the proxies is to gather data from the PRESTO sensors and to answer queries posed by the user. The PRESTO sensors are assumed to be battery-powered and remotely located. Their task is to sense the data and transmit it to PRESTO proxies, while archiving some of it locally on flash memory.

Now, let us discuss how PRESTO processes the sensor data acquisition query (Query 2.1). For answering such a query, the PRESTO proxies always maintain a time-series prediction model. Specifically, PRESTO maintains a seasonal ARIMA (SARIMA) model [120] of the following form for each sensor:

$$s_{ij} = s_{(i-1)j} + s_{(i-L)j} - s_{(i-L-1)j} + \Phi e_{i-1} - \Theta e_{i-L} + \Phi\Theta e_{i-L-1}, \tag{2.3}$$

where Φ and Θ are parameters of the SARIMA model, e_i are the prediction errors and L is known as the seasonal period. For example, while monitoring temperature, L could be set to one day, indicating that the current temperature (s_{ij}) is related to the temperature yesterday at the same time ($s_{(i-L)j}$) and a previous time instant ($s_{(i-L-1)j}$). In short, the seasonal period L allows us to model the periodicity that is inherent in certain types of data.

In the PRESTO system the proxies estimate the parameters of the model given in Eq. (2.3), and then transmit these parameters to individual PRESTO sensors. The PRESTO sensors use these models to predict the expected sensor value $\hat{s}_{ij}$, and only transmit the raw sensor value s_{ij} to the proxies when the absolute difference between the predicted expected sensor value and the raw sensor value is greater than a user-defined threshold δ. This task can be summarized as follows:

$$|s_{ij} - \hat{s}_{ij}| > \delta, \ \text{transmit } s_{ij} \text{ to proxy.} \tag{2.4}$$

The PRESTO proxy also provides a confidence interval for each predicted value it computes using the SARIMA model. Like BBQ (refer Section 2.2.2.2), this confidence interval can also be used for query processing, since it represents an error bound on the predicted sensor value. Similar to BBQ, a given PRESTO proxy will query the corresponding sensors only when the desired confidence interval, specified by the query, could not be satisfied with the values stored at that proxy. In most cases, the values stored at the proxy can be used for query processing, without acquiring any further values from the PRESTO sensors [82]. The only difference between PRESTO and BBQ is that, PRESTO uses a different measure of confidence as compared to BBQ. Further details of this confidence interval can be found in [82].

2.2.3.2 Ken

To reduce the communication cost, the Ken [29] framework employs a similar strategy as PRESTO. The key difference between Ken and PRESTO is that PRESTO uses a SARIMA model; which only takes into account temporal correlations. On the other hand, Ken uses a dynamic probabilistic model that takes into account spatial and temporal correlations in the data. Since a large quantity of sensor data is correlated spatially, and not only temporally, Ken derives advantage from such spatio-temporal correlation.

The Ken framework has two types of entities, *sink* and *source*. Their functionalities and capabilities are similar to the PRESTO proxy and the PRESTO sensor respectively. The only difference is that the PRESTO sensor only represents a single sensor, as opposed to a Ken source that could include multiple sensors or a sensor network. The sink is the base station to which the sensor values s_{ij} are communicated by the source (refer Figure 2.2).

The fundamental idea behind Ken is that both the source and the sink maintain the same dynamic probabilistic model of data evolution. The source only communicates with the sink when the raw sensor values deviate beyond a certain bound, as compared to the predictions from the dynamic probabilistic model. In the meantime, the sink uses the sensor values predicted by the model.

As discussed before, Ken uses a dynamic probabilistic model that considers spatio-temporal correlations. Particularly, its dynamic probabilistic model computes the following pdf at the source:

$$\begin{aligned}\mathbb{P}(R_{(i+1)1},\ldots,R_{(i+1)n}|\boldsymbol{r}_1,\ldots,\boldsymbol{r}_i) = \int & \mathbb{P}(R_{(i+1)1},\ldots,R_{(i+1)n}|R_{i1},\ldots,R_{in}) \\ & \mathbb{P}(R_{i1},\ldots,R_{in}|\boldsymbol{r}_1,\ldots,\boldsymbol{r}_i) dR_{i1}\ldots dR_{in}. \qquad (2.5)\end{aligned}$$

This pdf is computed using the observations that have been communicated to the sink; the values that are not communicated to the sink are ignored by the source, since they do not affect the model at the sink. Next, each sensor contained in the source computes the expected sensor value using Eq. (2.5) as follows:

$$\bar{s}_{(i+1)j} = \int R_{(i+1)j}\mathbb{P}(R_{(i+1)1},\ldots,R_{(i+1)n}) dR_{(i+1)1}\ldots dR_{(i+1)n}. \qquad (2.6)$$

The source does not communicate with the sink if $|\bar{s}_{(i+1)j} - s_{(i+1)j}| < \delta$, where δ is a user-defined threshold. If this condition is not satisfied, the source communicates to the sink the smallest number of sensor values, such that the δ threshold would be satisfied. Similarly, if the sink does not receive any sensor values from the source, it computes the expected sensor values $\bar{s}_{(i+1)j}$ and uses them as an approximation to the raw sensor values. If the sink receives a few sensor values form the source, then, before computing the expected values, the sink updates its dynamic probabilistic model.

2.2.3.3 A Generic Push-Based Approach

The last push-based approach that we will survey is a generalized version of other push-based approaches [75]. This approach is proposed by Silberstein *et al.* [121]. Like other push-based approaches, the base station and the sensor network agree on an expected behavior, and, as usual, the sensor network reports values only when there is a substantial deviation from the agreed behavior. But, unlike other approaches, the definition of expected behavior proposed in [121] is more generic, and is not limited to a threshold δ.

In this approach a sensor can either be an updater (one who acquires or forwards sensor values) or an observer (one who receives sensor values). A sensor node can be both, updater and observer, depending on whether it is on the boundary of the sensor network or an intermediate node. The updaters and the observers maintain a model encoding function f_{enc} and a decoding function f_{dec} . These model encoding/decoding functions define the agreed behavior of the sensor values. The updater uses the encoding function to encode the sensor value s_{ij} into a transmission message g_{ij} , and transmits it to the observer.

The observer uses the decoding function f_{dec} to decode the message g_{ij} and construct $\hat{s}_{ij}$. If the observer finds that s_{ij} has not changed significantly, as defined by the encoding function, then the observer transmits a `null` symbol. A `null` symbol indicates that the sensor value is *suppressed* by the observer. Following is an example of the encoding and decoding functions [121]:

$$f_{enc}(s_{ij}, s_{i'j}) = \begin{cases} g_{ij} = s_{ij} - s_{i'j}, & \text{if } |s_{ij} - s_{i'j}| > \delta; \\ g_{ij} = \texttt{null}, & \text{otherwise.} \end{cases} \tag{2.7}$$

$$f_{dec}(g_{ij}, \hat{s}_{(i-1)j}) = \begin{cases} \hat{s}_{(i-1)j} + g_{ij}, & \text{if } g_{ij} \neq \texttt{null}; \\ \hat{s}_{(i-1)j}, & \text{if } g_{ij} = \texttt{null}. \end{cases} \tag{2.8}$$

In the above example, the encoding function f_{enc} computes the difference between the model predicted sensor value $s_{i'j}$ and the raw sensor value s_{ij}. Then, this difference is transmitted to the observer only if it is greater than δ, otherwise the `null` symbol is transmitted. The decoding function f_{dec} decodes the sensor value $\hat{s}_{(i-1)j}$ using the message g_{ij}.

The encoding and decoding functions in the above example are purposefully chosen to demonstrate how the δ threshold approach can be replicated by these functions. More elaborate definitions of these functions, that may be used for encoding complicated behavior can be found in [121].

2.3 Model-Based Data Cleaning

A well-known characteristic of time-series data is that it is uncertain and erroneous. This is due to the fact that the data sources often operate with discharged batteries, network failures, and imprecision. Other factors, such as low-cost sensors, freezing or heating of the casing or measurement device, accumulation of dirt, mechanical failure or vandalism (from humans or animals) heavily affect the quality of the time-series data [47, 63, 138]. This may cause a significant problem with respect to data utilization, since applications using erroneous data may yield unsound results. For example, scientific applications that perform prediction tasks using observation data obtained from cheap and less-reliable sensors may produce inaccurate prediction results.

To address this problem, it is essential to detect and correct erroneous values in time-series data by employing *data cleaning*. The data cleaning task typically involves complex processing of data [62, 136]. In particular, it becomes more difficult for time-series data, since true data values corresponding to erroneous data values are generally unobservable. This has led to a new approach – *model-based data cleaning*. In this approach, the most probable data values are inferred using well-established models, and then anomalies are detected by comparing raw data values with the corresponding inferred data values. In the literature there are various suggestions for model-based approaches for data cleaning. This section describes the key mechanisms proposed by these approaches, particularly focusing on the models used in the data cleaning process.

2.3.1 Overview of the Data Cleaning System

A system for cleaning time-series data generally consists of four major components: *user interface, stream processing engine, anomaly detector*, and *data storage* (refer Figure 2.4). In the following, we describe each component.

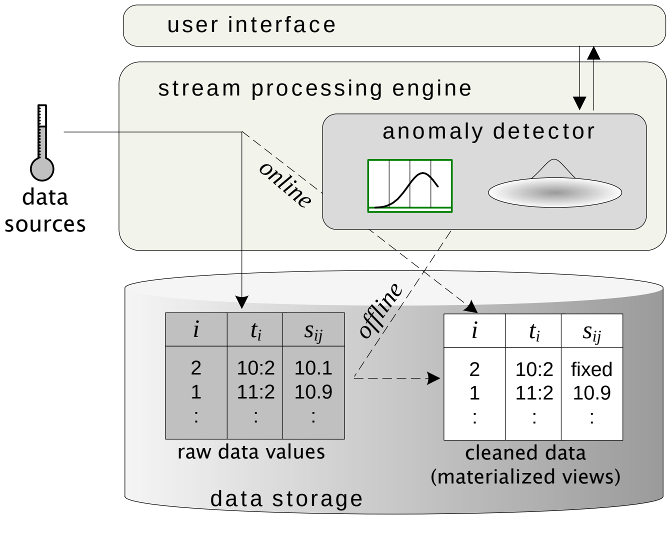


Figure 2.4: Architecture of time-series data cleaning system.

User Interface: The user interface plays two roles in the data cleaning process. First, it takes all necessary inputs from users to perform data cleaning, e.g., name of data source and parameter settings for models. Second, the results of data cleaning, such as 'dirty' data values captured by the anomaly detector, are presented using graphs and tables, so that users can confirm whether each candidate of such dirty values is an actual error. The confirmed results are then stored to (or removed from) the underlying data storage or materialized views.

Anomaly Detector: The anomaly detector is a core component in data cleaning. It uses models for detecting abnormal data values. The anomaly detector works in online as well as offline mode. In the online mode, whenever a new data value is delivered to the stream processing engine, the dirtiness of this value is investigated using various techniques and the errors are filtered out instantly. In the offline mode, the data is cleaned periodically, for instance, once per day. In the following subsections, we will review popular models used for online anomaly detection.

Stream Processing Engine: The stream processing engine maintains streaming data, while serving as a main platform where the other system components can cooperatively perform data cleaning. The anomaly detector is typically embedded into the stream processing engine, it may also be implemented as a built-in function on database systems.

Data Storage: The data storage maintains not only original data values, but also the corresponding cleaned data, typically in materialized views. This is because many applications often need to repeatedly perform data cleaning over the same data using different parameter settings for the models, especially when the previous parameter settings turn out to be inappropriate later. Therefore, it is important for the system to store cleaned data in database views without changing the original data, so that data cleaning can be performed again at any point of time (or time interval) as necessary.

2.3.2 Models for Data Cleaning

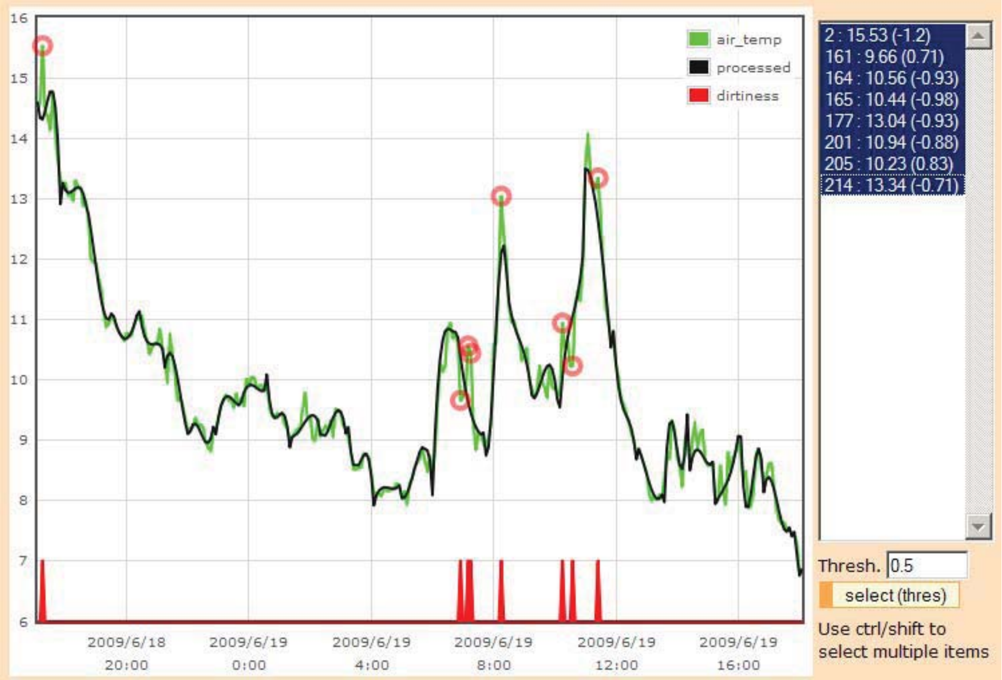
This subsection reviews popular models that are widely used in the data cleaning process.

2.3.2.1 Regression Models

Since time-series data values are a representation of physical processes, it is naturally possible to uncover the following properties: continuity of the sampling processes and correlations between different sampling processes. In principle, regression-based models exploit either or both of these properties. Specifically, they first compute the dependency from one variable (e.g., time) to another (e.g., data value), and then consider the regression curves as standards over which the inferred data values reside. The two most popular regression-based approaches use polynomial and Chebyshev regression for cleaning sensor values.

Polynomial Regression: Polynomial regression finds the best-fitting curve that minimizes the total difference between the curve and each raw data value s_{ij} at time t_i. Given a degree D, polynomial regression is formally defined as:

$$s_{ij} = \alpha_0 + \alpha_1 \cdot t_i + \cdots + \alpha_D \cdot t_i^D, \tag{2.9}$$



2 : 15.53 (-1.2)
161 : 9.66 (0.71)
164 : 10.56 (-0.93)
165 : 10.44 (-0.98)
177 : 13.04 (-0.93)
201 : 10.94 (-0.88)
205 : 10.23 (0.83)
214 : 13.34 (-0.71)

Thresh. 0.5

Use ctrl/shift to select multiple items

Figure 2.5: Detected anomalies based on a degree-2 Chebyshev regression.

where $\alpha_0, \alpha_1, \ldots, \alpha_D$ are regression coefficients.

Polynomial regression with high degrees approximates the given time series with more sophisticated curves, resulting in theoretically more accurate description of the raw data values. Practically, however, low-degree polynomials, such as constant ($D = 0$) and linear ($D = 1$), also perform satisfactorily. In addition, low-degree polynomials can be more efficiently constructed as compared to high-degree polynomials. A (weighted) moving average model [138] is also regarded as a polynomial regression.

Chebyshev Regression: Chebyshev regression is another popular model class for fitting data values, since they can quickly compute near-optimal approximations for a given time series. Suppose that time values t_i vary within a range $[\min(t_i), \max(t_i)]$. We, then, obtain normalized time values t'_i within a range $[-1, 1]$, by using the following transformation function $f(t_i)$ and its inverse transformation function $f^{-1}(t'_i)$ as follows:

$$f(t_i) = \left(t_i - \frac{\max(t_i) + \min(t_i)}{2}\right) \cdot \frac{2}{\max(t_i) - \min(t_i)}, \tag{2.10}$$

$$f^{-1}(t'_i) = \left(t'_i \cdot \frac{\max(t_i) - \min(t_i)}{2}\right) + \frac{\max(t_i) + \min(t_i)}{2}. \tag{2.11}$$

Next, given a degree D, a Chebyshev polynomial is defined as:

$$s_{ij} = f^{-1}(\cos(D \cdot \cos^{-1}(f(t_i)))).$$

Figure 2.5 illustrates a data cleaning process using degree-2 Chebyshev polynomials. Here, the raw sensor values are plotted as green curves, while the inferred values, obtained by fitting a Chebyshev polynomials, are overlaid by black curves. The anomaly points are then indicated by the underlying red histograms as well as red circles.

2.3.2.2 Probabilistic Models

In data cleaning, inferring data values is perhaps the most important task, since systems can then detect and clean dirty data values by comparing raw data values with the corresponding inferred data values. Figure 2.6 shows an example of the data cleaning process using probabilistic models. At time $t_i = 6$, the probabilistic model infers a probability distribution using the previous values $s_{2j}, \ldots, s_{5j}$ in the sliding window. The expected value $\bar{s}_{6j}$ (e.g., the mean of the Gaussian distribution in the future) is then considered as the inferred data value for data source w_j.

Next, the anomaly detector checks whether the raw data value s_{6j} resides within a reasonably accurate area. This is done in order to check whether the value is *normal.* For instance, the 3σ range can cover 99.7 % of the density in the figure, where s_{6j} is supposed to appear. Thus, the data cleaning process can consider that s_{6j} is not an error. At $t_i = 7$, the window slides and now contains raw sensor values $s_{3j}, \ldots, s_{6j}$. By repeating the same process, the anomaly detector finds s_{7j} resides out of the error bound (3σ range) in the inferred probability distribution, and is identified as an anomaly [116].

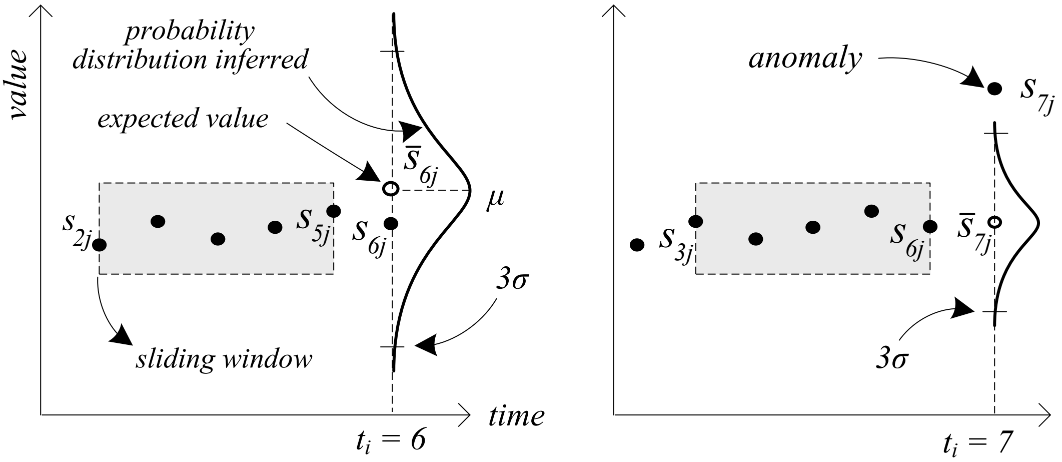


Figure 2.6: An example of data cleaning based on a probabilistic model.

A vast body of research work has utilized probabilistic models for computing inferred values. The *Kalman filter* is perhaps one of the most common probabilistic models to compute inferred values corresponding to raw sensor values. The Kalman filter is a stochastic and recursive data filtering algorithm that models the raw sensor value s_{ij} as a function of its previous value (or state) $s_{(i-1)j}$ as follows:

$$s_{ij} = \alpha_1 s_{(i-1)j} + \alpha_2 u_i + x_i,$$

where α_1 and α_2 are constants defining the state transition from time t_{i-1} to time t_i, x_i is the time-varying input at time t_i, and z_i is the process noise drawn from a zero mean multi-variate Gaussian distribution. In [123], the Kalman filter is used for detecting erroneous values, as well as inter/extrapolating missing sensor values. Jain *et al.* [60] also use the Kalman filter for filtering possible dirty values.

Similarly, Elnahrawy and Nath [44] proposed to use Bayes' theorem to estimate a probability distribution P_{ij} at time t_i from raw data values s_{ij}, and associate them with an error model, typically a normal distribution. Built on the same principle, a neuro-fuzzy regression model [105] and a belief propagation model based on Markov chains [30] were used to identify anomalies. Tran *et al.* [126] propose a method to infer missing or erroneous values in RFID data. All the techniques for inferring data values also enable quality-aware processing of sensor data streams [73, 74], since inferred data values can serve as the bases for indicating the quality or precision of the raw data values.

2.3.2.3 Outlier Detection Models

An *outlier* is a data value that largely deviates from the other data values. Obviously, outlier detection is closely related to the process of data cleaning. The outlier-detection techniques are well-categorized in the survey studies of [23, 104].

In particular, some of the outlier detection methods focus on sensor data [37, 119, 136]. Zhang *et al.* [136] offer an overview of such outlier detection techniques for sensor network applications. Deligiannakis *et al.* [37] consider correlation, extended Jaccard coefficients, and regression-based approximation for model-based data cleaning. Shen *et al.* [119] propose to use a histogram-based method to capture outliers. Subramaniam *et al.* [122] introduce distance- and density-based metrics that can identify outliers. In addition, the ORDEN system [47] detects polygonal outliers using the triangulated wireframe surface model.

2.3.3 Declarative Data Cleaning Approaches

From the perspective of using a data cleaning system, supporting a declarative interface is important since it allows users to easily control the system. This idea is reflected in a wide range of prior works that propose SQL-like interfaces for data cleaning [64, 91, 109]. These proposals hide complicated mechanisms of data processing or model utilization from the users, and facilitate data cleaning in time-series database.

```
DEFINE      [rule name]
ON          [table name]
FROM        [table name]
CLUSTER BY  [cluster key]
SEQUENCE BY [sequence key]
AS          [pattern]
WHERE       [condition]
ACTION      [DELETE | MODIFY | KEEP]
```

Figure 2.7: An example of anomaly detection using a SQL statement.

More specifically, Jeffery *et al.* [63, 64] divide the data cleaning process into five tasks: *Point, Smooth, Merge, Arbitrate*, and *Virtualize*. These tasks are then supported within a database system. For example, the SQL statement in Query 2.2 performs anomaly detection within a spatial granule by determining the average of the data values from different data sources in the same proximity group. Then, individual data values are rejected if they are outside of one standard deviation from the mean.

```
SELECT spatial_granule, AVG(temp)
FROM data s [Range By 5 min]
      (SELECT spatial_granule, avg(temp) as avg,
      stdev(temp) as stdev
      FROM data [Range By 5 min]) as a
WHERE a.spatial_granule = s.spatial_granule
      AND a.avg + (2*a.stdev) < s.temp
      AND a.avg - (2*a.stdev) > s.temp
```

Query 2.2: An example of anomaly detection using a SQL statement.

As another approach, Rao *et al.* [109] focus on a systemic solution, based on rewriting queries using a set of cleansing rules. Specifically, the system offers the rule grammar shown in Figure 2.7 to define and execute various data cleaning tasks. Unlike the prior relational database approaches, Mayfield *et al.* [91] model data as a graph consisting of nodes and links. They, then, provide an SQL-based, declarative framework that enables data owners to specify or discover groups of attributes that are correlated, and apply statistical methods that validate and clean the data values using such dependencies.

2.4 Model-Based Query Processing

In this section we elaborate another important task in time-series data management–query processing. We primarily focus on in-network and centralized query processing approaches. We consider different queries assuming the time-series database model described in Section 1.1, and then discuss how each approach processes these queries. In Section 2.2, however, we followed an approach where we chose a singe query (i.e., Query 2.1) and demonstrated how different techniques processed this query. On the contrary, in this section, we chose different queries for all the approaches, and then discuss these approaches along with the queries. We follow this procedure since, unlike Section 2.2, the assumptions made by each query processing technique are different. Thus, for highlighting the impact of these assumptions and simplifying the discussion, we select different queries for each approach.

2.4.1 In-Network Query Processing

In-network query processing first builds an overlay network (like, the SRT discussed in Section 2.2.2.1). Then, the overlay network is used for increasing the efficiency of aggregating data values and processing queries. For instance, while processing a threshold query, parent nodes send the query to the child nodes only when the query threshold condition overlaps with the range of data values contained in the child nodes, which is stored in the parent node's local memory.

Consider the threshold query given in Query 2.3. Query 2.3 requests the data source identifiers of all the data sources that have reported a temperature greater than 10°C at the current time instance. Before answering this query, we assume that we have already constructed a SRT as described in Section 2.2.1 (refer Figure 2.3). Query 2.3 is sent by the root node of the SRT to its children that are a part of the query response. The child nodes check whether the data value they have sensed is greater than 10°C. If the data value is greater than 10°C at a child node, then that child node appends its identifier to the query response. The child node, then, forwards the query to its children and waits for their response. Once all the children of a particular node have responded, then that node forwards the response of its entire sub-tree to its parent. In the end, the root node receives all the source identifiers w_j that have recorded temperature greater than 10°C.

`SELECT` w_j `FROM sensor_data WHERE` $s_{ij} > 10°\text{C}$ `AND` t_i `== NOW()`

Query 2.3: Return the sensor identifiers w_j where $s_{ij} > 10°\text{C}$.

2.4.2 Model-Based Views

The MauveDB [40] approach proposes standard database views [42] as an abstraction layer for processing queries. These views are maintained in a form of a regression model; thus they are called *model-based* views. The main advantage of this approach is that the model-based view can be incrementally updated as fresh data values are obtained from the data sources. Furthermore, incremental updates is an attractive feature, since such updates are computationally efficient.

Before processing any queries in MauveDB, we have to first create a model-based view. The query for creating a model-based view is shown in Query 2.4. The model-based view created by this query is called `RegModel`. `RegModel` is a regression model in which the temperature is the dependent variable and the position (x_j, y_j) is an independent variable (refer Figure 2.8). Note that `RegModel` is incrementally updated by MauveDB. At time t_1 values from sources w_1, w_3 and at time t_2 the value from data source w_2 are respectively used to update the view. The view update mechanism exploits the fact that regression functions can be updated. Further details regarding the update mechanism can be found in [40].

`CREATE VIEW` RegModel `AS FIT` s `OVER` x^2, xy, y^2, x, y `TRAINING_DATA SELECT` x_j, y_j, s_{ij}
`FROM sensor_data WHERE` $t_i > t_{start}$ `AND` $t_i < t_{end}$

Query 2.4: Model-based view creation query.

Once this step is performed many types of queries can be evaluated using the `RegModel` view. For instance, consider Query 2.5. MauveDB evaluates this query by interpolating the value of temperature at fixed intervals on the x- and y-axis; this is similar to database view materialization [42]. Then the positions (x, y) where the interpolated temperature value is greater than 10°C are returned.

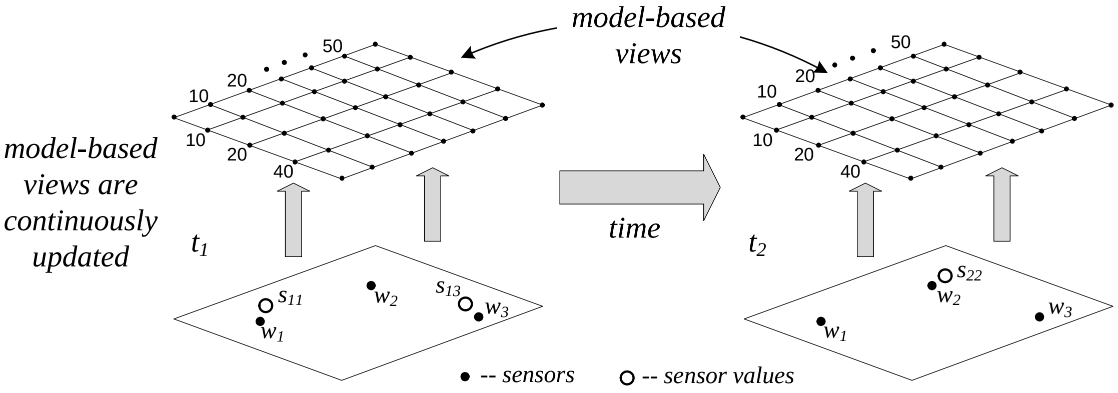


Figure 2.8: Example of the `RegModel` view with three sensors. `RegModel` is incrementally updated as new sensor values are acquired.

Admittedly, although updating the model-based view is efficient, but for processing queries the model-based view should be materialized at a certain fixed set of points. This procedure produces a large amount of overhead when the number of independent variables is large, since it dramatically increases the number of points where the view should be materialized.

`SELECT` x, y `FROM` RegModel `WHERE` $s > 10$°C

Query 2.5: Querying model-based views.

2.4.3 Symbolic Query Evaluation

This approach is proposed by the FunctionDB [125] system. FunctionDB, like MauveDB, also interpolates the values of the dependent variable, and then uses the interpolated values for query processing.

As discussed before, the main problem with value interpolation is that the number of points, where the data values should be interpolated, increase dramatically as a function of the number of independent variables. As a solution to this problem, FunctionDB symbolically executes the filter (for example, the `WHERE` clause in Query 2.5) and obtains feasible regions of the independent variables. These feasible regions are the regions that include the exact response to the query, at the same time contain a significantly low number of values to interpolate. FunctionDB evaluates the query by interpolating values only in the feasible regions, followed by a straightforward evaluation of the query.

FunctionDB treats the temperature reported by the data source w_j as a continuous function of time $f_j(t)$, instead of treating it as discrete values sampled at time stamps t_i. An example of a query in the FunctionDB framework is given in Query 2.6. This query returns the time values t between t_{start} and t_{end} where the temperature of the source w_1 is greater than 10°C. Note that the time values t are not necessarily the time stamps t_i where a particular sensor value was recorded.

`SELECT` t `WHERE` $f_1(t) > 10°\text{C}$ `AND` $t > t_{start}$ `AND` $t < t_{end}$ `GRID t 1s`

Query 2.6: Continuous threshold query.

For defining the values of the time axis t (or any continuous variable), FunctionDB proposes the `GRID` operator. The `GRID` operator specifies the interval at which the function $f_1(t)$ should be interpolated between time t_{start} and t_{end}. For instance, `GRID t 1s` indicates that the time axis should be interpolated at one second intervals between time t_{start} and t_{end}. To process Query 2.6, FunctionDB first symbolically executes the `WHERE` clause and obtains the feasible regions of the time axis (independent variable). Then, using the `GRID` operator, it generates time stamps $\mathcal{E}$ in the feasible regions. The data value is interpolated at the time stamps $\mathcal{E}$ using regression functions. Lastly, the query is processed on these interpolated values, and time stamps $\mathcal{E}' \subseteq \mathcal{E}$ where the temperature is greater than 10°C are returned.

2.4.4 Processing Queries over Uncertain Data

In this form of query processing the assumption is that the time-series data is inherently uncertain. This uncertainty is especially common with data originating from sensors. The various factors responsible for this uncertainty are: loss of calibration over time, faulty sensors, unsuitable environmental conditions, low accuracy, etc. Thus, the approaches that treat time-series data as uncertain, assume that each data value is associated with a random variable, and is drawn from a distribution. In this subsection, we discuss two such methods that model uncertain data by either a dynamic probabilistic model or a static probability distribution.

2.4.4.1 Dynamic Probabilistic Models

Dynamic probabilistic models (DPMs) are proposed for query processing in [60, 66]. These models continuously estimate a probability distribution. The estimated probability distribution is used for query processing. Mainly, there are two types of models that are frequently used for estimating dynamic probability distributions: particle filters and Kalman filters. Particle filters are generalized form of Kalman filters. Since we have already discussed Kalman filters in Section 2.3.2, here we will focus on particle filtering.

Consider a data source, say w_1. The particle filtering approach [13], at each time instant t_i, estimates and stores p weighted tuples $\{(o^1_{i1}, s^1_{i1}), \ldots, (o^p_{i1}, s^p_{i1})\}$, where the weight o^1_{i1} denotes the probability of s^1_{i1} being the data value of the data source w_1 at time t_i, and so on. An example of particle filtering is shown in the `pf_sensor_data` table in Figure 2.9.

Now, consider Query 2.7 that requests the average temperature `AVG`(s_{ij}) between time t_{start} and t_{end}. To evaluate this query, we assume that we already have executed the particle filtering algorithm at each time instance t_i and have created the `pf_sensor_data` table. We, then, perform the following two operations:

1. For each time t_i between t_{start} and t_{end}, we compute the expected temperature $\bar{s}_{i1} = \sum_{l=1}^{p} o_{i1}^{l} \cdot s_{i1}^{l}$. The formal SQL syntax for computing the expected values using the `pf_sensor_data` table is as follows:

   `SELECT` $t_i, \sum_{l=1}^{p} o_{i1}^{l} \cdot s_{i1}^{l}$ `FROM pf_sensor_data WHERE` $t_i > t_{start}$ `AND` $t_i < t_{end}$ `GROUP BY` t_i

2. The final result is the average of all the $\bar{s}_{i1}$ that we computed in Step 1.

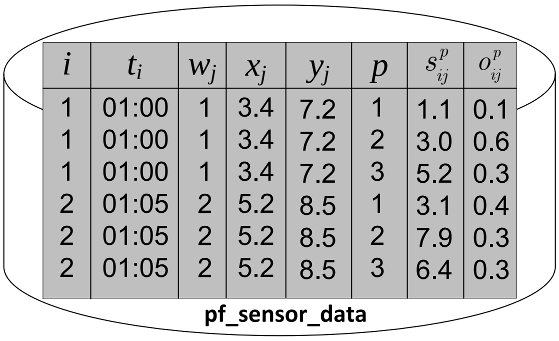
Essentially, the tuples $\{(o_{i1}^{1}, s_{i1}^{1}), \ldots, (o_{i1}^{p}, s_{i1}^{p})\}$ represent a discretized version of $\mathbb{P}(R_{i1})$, which is the pdf for the random variable R_{i1}. The most challenging tasks in particle filtering are to continuously infer weights $o_{i1}^{1}, \ldots, o_{i1}^{p}$ and to select the optimal number of particles p, keeping in mind a particular scenario and type of data [13].

`SELECT AVG`(s_{i1}) `FROM pf_sensor_data WHERE` $t > t_{start}$ `AND` $t < t_{end}$

Query 2.7: Compute the average temperature between time t_{start} and t_{end}.

2.4.4.2 Static Probabilistic Models

Cheng *et al.* [24, 25, 26] model the data value as obtained from an user-defined uncertainty range. For example, if the value of a temperature sensor is 15°C, then the actual value could vary between 13°C and 17°C. Furthermore, the assumption is that the data value is drawn from a static probability distribution that has support over the uncertainty range.



i	t_i	w_j	x_j	y_j	p	s_{ij}^{p}	o_{ij}^{p}
1	01:00	1	3.4	7.2	1	1.1	0.1
1	01:00	1	3.4	7.2	2	3.0	0.6
1	01:00	1	3.4	7.2	3	5.2	0.3
2	01:05	2	5.2	8.5	1	3.1	0.4
2	01:05	2	5.2	8.5	2	7.9	0.3
2	01:05	2	5.2	8.5	3	6.4	0.3

pf_sensor_data

Figure 2.9: Particle filtering stores p weighted data values for each time instance t_i.

Thus, for each source w_j we associate an uncertainty range between $\max(s_j)$ and $\min(s_j)$, in which the actual sensor values can be found. In addition, the pdf of the sensor values of sensor s_j is denoted as $\mathbb{P}_{ij}(s)$. Note that the pdf has non-zero support only between $\min(s_j)$ and $\max(s_j)$. Consider a query that requests the average temperature of the sensors w_1 and w_2 at time t_i. Since the values of the sources w_1 and w_2 are uncertain in nature, the response to this query is a pdf, denoted as $\mathbb{P}_{avg}(s)$. This pdf gives us the probability of the sensor value s being the average. $\mathbb{P}_{avg}(s)$ is computed using the following formula:

$$\mathbb{P}_{avg}(s) = \int_{\max(\min(s_1),s-\max(s_2))}^{\min(\max(s_1),s-\min(s_2))} \mathbb{P}_{i1}(y)\mathbb{P}_{i2}(s-x)dx. \tag{2.12}$$

Naturally, Eq. (2.12) becomes more complicated when there are many (and not only two) sensors involved in the query. Additional details about handling such scenarios can be found in [24].

2.4.5 Query Processing over Semantic States

The MIST framework [14] proposes to use Hidden Markov Models (HMMs) for deriving semantic meaning from the time-series data. HMMs allow us to capture the hidden states, which are sometimes of more interest than the actual time-series data. Consider, as an example, a scenario where the data sources $\mathcal{W}$ are used to monitor the temperature in all the rooms of a building. Generally, we are only interested to know which rooms are hot or cold, rather than the actual temperature in those rooms. Here, we can use a two-state HMM with states *Hot* and *Cold* to continuously infer the semantic states of the temperature in all the rooms.

Furthermore, MIST proposes an in-network index structure for indexing the HMMs. This index can be used for improving the performance of query processing. For instance, if we are interested in finding the rooms that are *Hot* with probability greater than 0.9, then the in-network model index can efficiently prune the rooms that are surely not a part of the query response. Here, we shall not cover the details of index construction and pruning. We encourage the interested reader to read the following paper [14].

2.4.6 Processing Event Queries

Event queries are another important class of queries that are proposed in the literature. These queries continuously monitor for a particular event that could probably occur in streaming time-series data. Consider a setup consisting of RFID sensors in a building. An event query could monitor an event of a person entering a room or taking coffee, etc. Event queries can also be registered, not only to monitor a single event, but a sequence of events that are important to a user. Again, we shall not cover any of the event query processing approaches in detail. The interested reader is referred to the prior works on this subject [89, 110, 126, 131].

2.5 Model-Based Data Compression

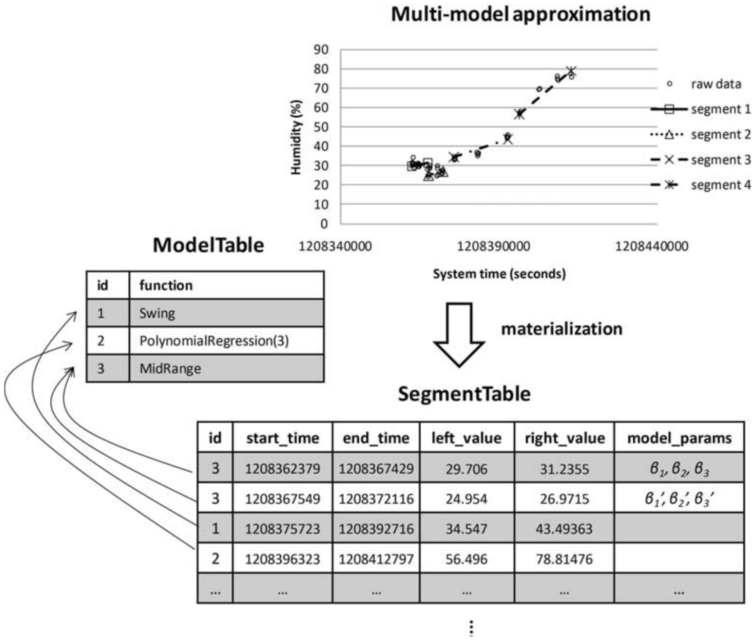
Recent advances in technology has resulted in the availability of a multitude of (often privately-held) sensors. Embedded sensing functionality (e.g., sound, accelerometer, temperature, GPS, RFID, etc.) is now included in mobile devices, like, phones, cars, or buses. The large number of these devices and the huge volume of raw monitored time-series data pose new challenges for sustainable storage and efficient retrieval of the time-series data streams. To this end, a multitude of model-based regression, transformation and filtering techniques have been proposed for approximation of time-series data streams. This section categorizes and reviews the most important model-based approaches for compression of time-series data. These models often exploit spatio-temporal correlations within data streams to compress the data within a certain error norm; this is also known as *lossy compression*. Several standard orthogonal transformation methods (like, Fourier or wavelet transform) reduce the amount of storage space required by reducing the dimensionality of data.

Unlike the assumptions of Section 2.2, where we assumed a time-series database consisting of several data sources, here we assume that we only have a single data source. We have dropped the several data sources assumption to simplify the notation and discussion in this section. Furthermore, we assume that the data values from the single data source are in a form of a *data stream*. Let us denote such a data stream as a sequence of data tuples (t_i, s_i), where s_i is the data value at time t_i.

2.5.1 Overview of Data Compression System

The goal of the data compression system is to approximate a data stream by a set of functions. Data compression methods that we are going to study in this section permit the occurrence of approximation errors. These errors are characterized by a specific error norm. A standard approach to sensor data compression is to segment the data stream into *data segments*, and then approximate each data segment, so that a specific error norm is satisfied. For example, if we are considering the L_∞ norm, then each data value of the data stream is approximated within an error bound ϵ .

Let us assume that we have K segments of a data stream. We denote these segments as $\mathcal{G}^1, \mathcal{G}^2, \ldots, \mathcal{G}^K$, where $\mathcal{G}^1$ approximates the data tuples $((t_1, s_1), \ldots, (t_{i_1}, s_{i_1}))$, while $\mathcal{G}^k$, where $k = 2, \ldots, K$, approximates the data items $((t_{i_{k-1}+1}, s_{i_{k-1}+1}), (t_{i_{k-1}+2}, s_{i_{k-1}+1}), \ldots, (t_{i_k}, s_{i_k}))$. Similar to [43], we distinguish between two classes of the segments used for approximation, namely *connected segments* and *disconnected segments*. In connected segments, the ending point of the previous segment is the starting point of the new segment. On the contrary, in disconnected segments, the approximation of the new segment starts from the subsequent data item in the stream. Disconnected segments offer more approximation flexibility and may lead to fewer segments; however, for linear approximation [71], they necessitate the storage of two data tuples (i.e., start tuple and end tuple) per data segment, as opposed to connected segments.



ModelTable

id	function
1	Swing
2	PolynomialRegression(3)
3	MidRange

SegmentTable

id	start_time	end_time	left_value	right_value	model_params
3	1208362379	1208367429	29.706	31.2355	$\theta_1, \theta_2, \theta_3$
3	1208367549	1208372116	24.954	26.9715	$\theta_1', \theta_2', \theta_3'$
1	1208375723	1208392716	34.547	43.49363	
2	1208396323	1208412797	56.496	78.81476	
...	...	...	...	...	...

Figure 2.10: The database schema for multi-model materialization.

Since functions are employed for approximating data segments, only the approximated data segments are stored in the database, instead of the raw data values of the data stream [103, 125]. A schema for linear segments is presented in [125], consisting of a table, referred to as `FunctionTable`, where each row represents a linear model with attributes `start_time`, `end_time`, `slope` and `intercept` (i.e., base) of the segment. In case of connected segments [43], the `end_time` attribute can be omitted.

A more generic schema for storing data streams, approximated by multiple models was proposed in [103] that consists of one table (`SegmentTable`) for storing the data segments, and a second table (`ModelTable`) for storing the model functions, as depicted in Figure 2.10. A tuple of the `SegmentTable` contains the approximation data for a segment in the time interval `[start_time, end_time]`. The attribute `id` stands for identification of the model that is used in the segment. The primary key in the `SegmentTable` is the `start_time`, while in the `ModelTable` it is `id`. When, both, linear and non-linear models are employed for approximation, `left_value` is the lowest raw data value encountered in the segment, and `right_value` is the highest raw data value encountered in the segment. In this case, `start_time`, `end_time`, `left_value` and `right_value` define a rectangular bucket that contains the values of the segment.

The attribute `model_params` stores the parameters of the model associated with the model identifier `id`. For example, regression coefficients are stored for the regression model. The attribute `model_params` has variable length (e.g., `VARCHAR` or `VARBINARY` data types in SQL) and it stores the concatenation of the parameters or their compressed representation, by means of standard lossless compression techniques (refer Section 2.5.7) or by a bitmap coding of approximate values, as proposed in [12]. Each tuple in the

`ModelTable` corresponds to a model with a particular `id` and `function`. The attribute `function` represents the name of the model and it maps to the names of two user defined functions (UDFs) stored in the database. The first function implements the mathematical formula of the model, and the second function implements the inverse mathematical formula of the model, if any. Both the UDFs are employed for answering value-based queries. While the first function is used for value regeneration over fixed time steps (also referred to as *gridding*), the second function is used for solving equations.

2.5.2 Methods for Data Segmentation

In [70], the piecewise linear approximation algorithms are categorized in three groups: sliding window, top-down and bottom-up. The sliding window approach expands the data segment as long as the data tuples fit. The bottom-up approach first applies basic data segmentation employing the sliding window approach. Then, for two consecutive segments, it calculates merging cost in terms of an approximation error. Subsequently, it merges the segments with the minimum cost within the maximum allowed approximation error, and updates the merging costs of the updated segments. The process ends when no further merging can be done without violating the maximum approximation error. The top-down approach recursively splits the stream into two segments, so as to obtain longest segments with the lowest error until all segments are approximated within the maximum allowed error.

Among these three groups, only the sliding window approach can be used online, but it employs look-ahead. The other two approaches perform better than the sliding window approach, but they need to scan all data, hence they cannot be used for approximating streaming data. Based on this observation, Keogh *et al.* [70] propose a new algorithm that combines the online processing property of the sliding window approach and the performance of the bottom-up approach. This approach needs a predefined buffer length. If the buffer is small, then it may produce many small data segments; if the buffer is large, then there is a delay in returning the approximated data segment. The maximum look-ahead size is constrained by the maximum allowed delay between data production and data reporting or data archiving.

2.5.3 Piecewise Approximation

Among several different data stream approximation techniques, piecewise linear approximation has been the most widely used [70, 78]. Piecewise linear approximation models the data stream with a separate linear function per data segment. Piecewise constant approximation (PCA) approximates a data segment with a constant value, which can be the first value of the segment (referred to as the cache filter) [99], the mean value or the median value (referred to as poor man's compression - midrange (PMC-MR) [78]).

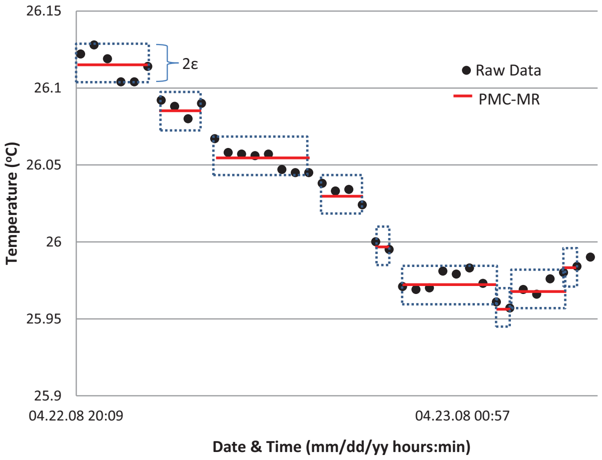


Figure 2.11: Poor Man's Compression - MidRange (PMC-MR).

In the cache filter, for all the sensor values in a segment $\mathcal{G}^k$, the following condition should be satisfied:

$$\left|s_{i_{k-1}+h} - s_{i_{k-1}+1}\right| < \epsilon \qquad \text{for } h = 1, \ldots, i_k, \tag{2.13}$$

where ϵ is the maximum allowed approximation error according to the L_∞ norm. Also, for PMC-Mean and PMC-MR the sensor values in a segment g^k should satisfy the following condition:

$$\max_{1 \le h \le i_k} s_{i_{k-1}+h} - \min_{1 \le h \le i_k} s_{i_{k-1}+h} \le 2\epsilon\ . \tag{2.14}$$

Furthermore, for PMC-Mean, the approximation value for the segment $\mathcal{G}^k$ is given by the mean value of the sensor values in segment $\mathcal{G}^k$. But, for PMC-MR it is given as follows:

$$\frac{\max_{1 \le h \le i_k} s_{i_{k-1}+h} - \min_{1 \le h \le i_k} s_{i_{k-1}+h}}{2}\ .$$

The data segmentation approach for PMC-MR is illustrated in Figure 2.11.

Moreover, the linear filter [70] is a simple piecewise linear approximation technique in which the data values are approximated by a line connecting the first and second point of the segment. When a new data tuple cannot be approximated by this line with the specified error bound, a new segment is started. In [43], two new piecewise linear approximation models were proposed, namely *Swing* and *Slide*, that achieve much higher compression compared to the cache and linear filters. We briefly discuss the swing and slide filters below.

2.5.3.1 Swing and Slide Filters

The swing filter is capable of approximating multi-dimensional data but, for simplicity, we describe its algorithm for one-dimensional data. Given the arrival of two data tuples (t_1, s_1) and (t_2, s_2) of the first segment of the data stream, the swing filter maintains a set of lines, bounded by an upper line $\max(s_1)$ and a lower line $\min(s_1)$. $\max(s_1)$ is defined by the pair of points (t_1, s_1) and $(t_2, s_2 + \epsilon)$, while $\min(s_1)$ is defined by the pair of points (t_1, s_1) and $(t_2, s_2 - \epsilon)$, where ϵ is the maximum approximation error bound. Any line segment between $\max(s_1)$ and $\min(s_1)$ can represent the first two data tuples. When (t_3, s_3) arrives, first it is checked whether it falls within the lines $\min(s_1)$, $\max(s_1)$. Then, in order to maintain the invariant that all lines within the set can represent all data tuples so far, $\min(s_1)$ (respectively $\max(s_1)$) may have to be adjusted to the higher-slope (respectively lower-slope) line defined by the pair of data tuples $((t_1, s_1), (t_3, s_3 - \epsilon))$ (respectively $((t_1, s_1), (t_3, s_3 + \epsilon))$). Lines below this new $\min(s_1)$ or above this new $\max(s_1)$ cannot represent the data tuple (t_3, s_3). The segment estimation continues until the new data tuple falls out of the upper and lower lines for a segment. The generated line segment for the completed filtering interval is chosen so as to minimize the mean square error for the data tuples observed in that interval. As opposed to the slide filter (described below), in the swing filter the new data segment starts from the end point of the previous data segment.

In the slide filter, the operation is similar to the swing filter, but upper and lower lines u and b are defined differently. Specifically, after (t_1, s_1) and (t_2, s_2) arrive, $\max(s_1)$ is defined by the pair of data tuples $(t_1, s_1 - \epsilon)$ and $(t_2, s_2 + \epsilon)$, while $\min(s_1)$ is defined by $(t_1, s_1 + \epsilon)$ and $(t_2, s_2 - \epsilon)$. After the arrival of (t_3, s_3), $\min(s_1)$ (respectively $\max(s_1)$) may need to be adjusted to the higher-slope (respectively lower-slope) line defined by $((t_j, s_j + \epsilon), (t_3, s_3 - \epsilon))$ (respectively $((t_i, s_i - \epsilon), (t_3, s_3 + \epsilon))$), where $i \in [1, 2]$. The slide filter also includes a look-ahead of one segment, in order to produce connected segments instead of disconnected segments, when possible.

Palpanas *et al.* [101] employ *amnesic functions* and propose novel techniques that are applicable to a wide range of user-defined approximating functions. According to amnesic functions, recent data is approximated with higher accuracy, while higher error can be tolerated for older data. Yi and Faloutsos [135] suggested approximating a data stream by dividing it into equal-length segments and recording the mean value of the sensor values that fall within the segment (referred to as segmented means or as piecewise aggregate approximation (PAA)). On the other hand, adaptive piecewise constant approximation (APCA) [21] allows segments to have arbitrary lengths.

2.5.3.2 Piecewise Linear Approximation

The piecewise linear approximation uses the linear regression model for compressing data streams. The linear regression model of a data segment is given as:

$$s_i = \alpha_0 \cdot t_i + \alpha_1, \tag{2.15}$$

where α_0 and α_1 are known as the slope and the base respectively. The difference between s_i and t_i is known as the residual for time t_i. For fitting a linear regression model of Eq. (2.15) to the data values $s_i : t_i \in [\max(t_i); \min(t_i)]$, the ordinary least squares (OLS) estimator is employed. The OLS estimator selects α_0 and α_1, such that they minimize the following sum of squared residuals:

$$RSS(\alpha_0, \alpha_1) = \sum_{t_i=\min(t_i)}^{\max(t_i)} [s_i - (\alpha_0 \cdot t_i + \alpha_1)]^2.$$

Therefore, α_0 and α_1 are given as:

$$\begin{aligned} \alpha_1 &= \sum_{t_i=\min(t_i)}^{\max(t_i)} \left(\frac{t_i - \frac{\min(t_i)+\max(t_i)}{2}}{\sum_{t_i=\min(t_i)}^{\max(t_i)} (t_i - \frac{\min(t_i)+\max(t_i)}{2}) t_i} \right) s_i, \\ \alpha_0 &= \frac{\sum_{t_i=\min(t_i)}^{\max(t_i)} s_i}{\min(t_i) - \max(t_i) + 1} - \alpha_1 \frac{\min(t_i) + \max(t_i)}{2}. \end{aligned} \tag{2.16}$$

Here, the storage record of each data segment of the data stream consists of ([$\min(t_i)$, $\max(t_i)$]; α_0, α_1), where [$\min(t_i)$; $\max(t_i)$] is the segment interval, and α_0 and α_1 are the slope and base of the linear regression, as obtained from Eq. (2.16).

Similarly, instead of the linear regression model, a polynomial regression model (refer Eq. (2.9)) can also be utilized for approximating each segment of the data stream. The storage record of the polynomial regression model is similar to the linear regression model. The only difference is that for the polynomial regression model the storage record contains parameters $\alpha_1, \ldots, \alpha_D$ instead of the parameters α_0 and α_1.

2.5.4 Compressing Correlated Data Streams

Several approaches [36, 48, 84] exploit correlations among different data streams for compression. The GAMPS approach [48] dynamically identifies and exploits correlations among different data segments and then jointly compresses them within an error bound employing a polynomial-time approximation algorithm. In the first phase, data segments are individually approximated based on piecewise constant approximation (specifically the PMC-Mean described in Section 2.5.3). In the second phase, each data segment is approximated by a ratio with respect to a base segment. The segment formed by the ratios is called the ratio segment. GAMPS proposes to store the base segment and the ratio segment, instead of storing the original data segment. The idea here is that, in practice, the ratio segment is flat and therefore can be significantly compressed as compared to the original data segment.

The objective of the GAMPS approach is to identify a set of base segments, and associate every data segment with a base segment, such that the ratio segment can be used for reconstructing the data segment within a L_∞ error bound. The problem of identification of the base segments is posed as a *facility location* problem. Since this problem is NP-hard, a polynomial-time approximation algorithm is used for solving it, and producing the base segments and the assignment between the base segments and data segments.

Prior to GAMPS, Deligiannakis *et al.* [36] proposed the self-based regression (SBR) algorithm that also finds a base-signal for compressing historical sensor data based on spatial correlations among different data streams. The base-signal for each segment captures the prominent features of the other signals, and SBR finds piecewise correlations (based on linear regression) to the base-signal. Lin *et al.* [84] proposed an algorithm, referred to as adaptive linear vector quantization (ALVQ), which improves SBR in two ways: (i) it increases the precision of compression, and (ii) it reduces the bandwidth consumption by compressing the update of the base signal.

2.5.5 Multi-Model Data Compression

The potential burstiness of the data streams and the error introduced by the data sources often result in limited effectiveness of a single model for approximating a data stream within the prescribed error bound. Acknowledging this, Lazaridis *et al.* [78] argue that a global approximation model may not be the best approach and mention the potential need for using multiple models. In [79], it is also recognized that different approximation models are more appropriate for data streams of different statistical properties. The approach in [79] aims to find the best model approximating the data stream based on the overall *hit ratio* (i.e., the ratio of the number of data tuples fitting the model to the total number of data tuples).

Papaioannou *et al.* [103] aim to effectively find the best combination of different models for approximating various segments of the stream regardless of the error norm. They argue that the selection of the most efficient model depends on the characteristics of the data stream, namely rate, burstiness, data range, etc., which cannot be always known *a priori* for some data sources (e.g., sensors) and they can even be dynamic. Their approach dynamically adapts to the properties of the data stream and approximates each data segment with the most suitable model. They propose a greedy approach in which they employ multiple models for each segment of the data stream and store the model that achieves the highest compression ratio for the segment. They experimentally proved that their multi-model approximation approach always produces fewer or equal data segments than those of the best individual model. Their approach could also be used to exploit spatial correlations among different attributes from the same location, e.g., humidity and temperature from the same stationary sensor.

2.5.6 Orthogonal Transformations

The main application of the orthogonal transformation approaches has been in dimensionality reduction, since reducing the dimensionality improves performance of indexing techniques for similarity search in large collections of data streams. Typically, sequences of fixed length are mapped to points in a multi-dimensional Euclidean space; then, multi-dimensional access methods, such as R-tree family, can be used for fast access of those points. Since sequences are usually long, a straightforward application of the above approach, which does not use dimensionality reduction, suffers from performance degradation due to the curse of dimensionality [112].

The process of dimensionality reduction can be described as follows. The original data stream or signal is a finite sequence of real values or coefficients, recorded over time. This signal is transformed (using a specific transformation function) into a signal in a transformed space. To achieve dimensionality reduction, a subset of the coefficients of the orthogonal transformation are selected as features. These features form a feature space, which is simply a projection of the transformed space. The basic idea is to approximate the original data stream with a few coefficients of the orthogonal transformation; thus reducing the dimensionality of the data stream.

2.5.6.1 Discrete Fourier Transform (DFT)

The Fourier transform is the most popular orthogonal transformation. It is based on the simple observation that every signal can be represented by a superposition of sine and cosine functions. The discrete Fourier transform (DFT) and discrete cosine transform (DCT) are efficient forms of the Fourier transform often used in applications [6, 46]. The Discrete Fourier Transform of a time sequence $x = x_0, \ldots, x_{N-1}$ is a sequence $X = X_0, \ldots, X_{N-1}$ of complex numbers given by:

$$X_k = \sum_{j=0}^{N-1} e^{-i2\pi \frac{k}{N} j}. \tag{2.17}$$

The original signal can be reconstructed by the inverse Fourier transform of X, which is given by:

$$x_j = \sum_{k=0}^{N-1} X_k e^{i2\pi \frac{k}{N} j}. \tag{2.18}$$

In [6], Agrawal *et al.* suggest using the DFT for dimensionality reduction of long observation sequences. They argue that most real signals only require a few DFT coefficients for their approximation. Thus similarity search can be performed only over the first few DFT coefficients, instead of the full observation sequence. This provides an efficient approximate solution to the problem of similarity search in high-dimensional spaces. They use the Euclidean distance as the dissimilarity measure.

StatStream [137] is proposed for monitoring a large number of streams in real time. It employs a sliding window that is subdivided into a fixed number of basic windows. It also maintains DFT coefficients for each basic window. This allows a batch update of DFT coefficients over the entire history. StatStream uses the first $\hat{n}$, $\hat{n} \leq 2n$, dimensions of the DFT feature space for indexing. They superimpose an $\hat{n}$-dimensional orthogonal regular grid on the DFT feature space and partition it into cells with the same size and shape. Each data stream is mapped to a cell, based on its first $\hat{n}$ normalized DFT coefficients. Proximity of these first $\hat{n}$ normalized DFT coefficients is employed to report correlations.

Wavelets are another important class of orthogonal transformation. Wavelets can be thought of as a generalization of the Fourier transform to a much larger family of functions than sine and cosine. While Fourier transform has a single basis function (the exponential function), wavelets use an infinite family of basis functions. Ganesan *et al.* [49, 50] proposed in-network storage of wavelet-based summaries of time-series data. Recently, discrete wavelet transform (DWT) was also proposed in [22, 107] for data compression. For sustainable storage and querying, they propose progressive aging of summaries and load sharing techniques.

2.5.7 Lossless vs. Lossy Compression

While lossless compression is able to accurately reconstruct the original data, lossy compression techniques approximate data streams within a certain error bound. Most lossless compression schemes perform two steps in sequence: the first step generates a statistical model for the input data, and the second step uses this model to map input data to bit sequences. These bit sequences are mapped in such a way that frequently encountered data will produce shorter output than infrequent data. General-purpose compression schemes include DEFLATE (employed by gzip, ZIP, PNG, etc.), LZW (employed by GIF, compress, etc.), LZMA (employed by 7zip). The primary encoding algorithms used to produce bit sequences are Huffman coding (also used by DEFLATE) and arithmetic coding. Arithmetic coding achieves compression rates close to the best possible, for a particular statistical model, which is given by the information entropy. On the other hand, Huffman compression is simpler and faster but produces poor results.

Lossless compression techniques, however, are not adequate for a number of reasons: (a) as experimentally found in [78], gzip lossless compression achieves poor compression (50%) compared to lossy techniques, (b) lossless compression and decompression are usually more computationally intensive than lossy techniques, and (c) indexing cannot be employed for archived data with lossless compression.

2.6 Summary

In this chapter, we presented a comprehensive overview of the various aspects of model-based time-series data management. We surveyed the model-based techniques for data acquisition, handling missing data, outlier detection, data compression, data aggregation and summarization. We started with acquisition techniques like TinyDB [89], Ken [29], PRESTO [82]. In particular, we focused on how acqusitional queries are disseminated in the sensor network using routing trees [88]. Then we surveyed various approaches for time-series data cleaning, including polynomial-based [138], probabilistic [44, 105, 123, 126] and declarative [63, 91].

For processing spatial, temporal and threshold queries, we detailed query processing approaches like MauveDB [40], FunctionDB [125], particle filtering [66], MIST [14], etc. Here, our primary objective was to demonstrate how model-based techniques are used for improving various aspects of querying time-series data. Lastly, we discussed data compression techniques, like, linear approximation [70, 78, 101], multi-model approximations [78, 79, 103] and orthogonal transformations [6, 22, 46, 107].

All the methods that we presented in this chapter were model-based. They utilized models – statistical or otherwise – for describing, simplifying or abstracting various components of time-series data management. In the next chapter, keeping in mind the challenge of data scale, we will present the AFFINITY framework for querying statistical measures on time-series data.

Chapter 3

Affinity: Efficiently Querying Statistical Measures on Time-Series Data

> *To be a good mathematician, or a good gambler, or good at anything, you must be a good guesser. In order to be a good guesser, you should be, I would think, naturally clever to begin with. Yet to be naturally clever is certainly not enough. You should examine your guesses, compare them with the facts, modify them if need be, and so acquire an extensive (and intensive) experience with guesses that failed and guesses that came true. With such an experience in your background, you may be able to judge more competently which guesses have a chance to turn out correct and which have not.*
>
> George Pólya
> *How to Solve it*, 1954

3.1 Introduction

In the recent years we are experiencing a dramatic increase in the amount of available time-series data. A typical processing need of large-scale time-series data is statistical querying and mining in order to analyze trends and detect interesting correlations. In this chapter, we propose the Affinity framework that supports efficient processing of statistical queries on large time-series databases, based on the use of *affine relationships* among different time series. Before rigorously developing the technical approaches, let us, in the following, introduce the concept of affine relationships and motivate why they are a powerful tool to improve efficiency of statistical querying over time-series data.

Computing statistical measures.

An important challenge concerning time-series data processing is computing and storing statistical measures. For example, the correlation coefficient is a frequently used statistical measure for financial data. It is well-known that stock traders and investors are interested to find the correlation coefficient among pairs of stocks. Specifically, traders are interested in solving the following problem [16, 18, 58, 118]:

PROBLEM 3.1: Given the intra-day stock quotes of n stocks obtained at a sampling interval Δt, return the correlation coefficients of the $\frac{n(n-1)}{2}$ pairs of stocks on a given day.

As an example, let us consider daily time series of three stocks (i.e., $n = 3$), Intel Corporation (INTC), Advanced Micro Devices (AMD) and Microsoft Corporation (MSFT) on 2^{nd} January 2003 (refer Figure 3.1). Let us denote the stock price at time i of INTC, AMD and MSFT as s_{i1}, s_{i2} and s_{i3} respectively where $1 \leq i \leq m$. Using the integers 1, 2, and 3 to identify the time series $\boldsymbol{s}_1$, $\boldsymbol{s}_2$ and $\boldsymbol{s}_3$[1], we can form three pairs of the time series: $(1, 2)$, $(2, 3)$ and $(1, 3)$. A naive approach for solving Problem 3.1 is to compute the correlation coefficients for all the pairs of stocks for the day specified by the problem. Clearly, for high values of n this method does not scale well, since it computes the correlation coefficient for all the $\frac{n(n-1)}{2}$ pairs from scratch.

The first idea that this chapter proposes in order to enhance the naive approach is to exploit *affine relationships* between pairs of time-series data. For every $1 \leq i \leq m$, an affine relationship between pairs $(1, 3)$ and $(2, 3)$ can be defined by using an affine

---

[1] $\boldsymbol{s}_1 = (s_{11}, s_{21}, \ldots, s_{m1})$ is a vector of size m-by-1. Similarly for $\boldsymbol{s}_2$ and $\boldsymbol{s}_3$.

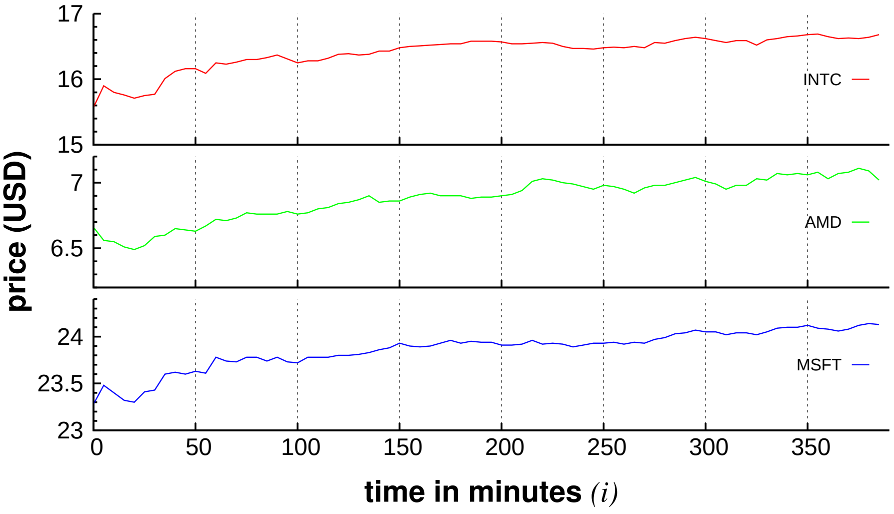


Figure 3.1: Stock prices for symbols INTC, AMD and MSFT on 2^{nd} January 2003.

transformation:

$$\begin{aligned}\begin{pmatrix} s_{i2} \\ s_{i3} \end{pmatrix} &= \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \begin{pmatrix} s_{i1} \\ s_{i3} \end{pmatrix} + \begin{pmatrix} b_1 \\ b_2 \end{pmatrix}, \\ &= \mathbf{A}_e \begin{pmatrix} s_{i1} \\ s_{i3} \end{pmatrix} + \boldsymbol{b}_e. \end{aligned} \tag{3.1}$$

The matrix $\mathbf{A}_e$ is known as the *transformation matrix* and the vector $\boldsymbol{b}_e$ represents a translation. Let us assume for the moment that the relationship between pairs of time series can be described at all time instants i using the same affine relationship. Obviously, as it can be seen from Figure 3.1, this is not true, but we will deal with this issue subsequently. Then, given the affine relationship in Eq. (3.1), the correlation coefficient between a *related* pair $(2, 3)$ could be computed directly from the correlation coefficient between pair $(1, 3)$, without accessing the time series. Specifically, consider the covariance matrix for the pair $(1, 3)$ denoted as Σ_{13} and defined as:

$$\Sigma_{13} = \begin{pmatrix} \sigma_1^2 & \rho_{13}\sigma_1\sigma_3 \\ \rho_{13}\sigma_1\sigma_3 & \sigma_3^2 \end{pmatrix}, \tag{3.2}$$

where ρ_{13} denotes the correlation coefficient between time series $\boldsymbol{s}_1$ and $\boldsymbol{s}_3$, similarly σ_1^2 and σ_3^2 are the variances of the time series $\boldsymbol{s}_1$ and $\boldsymbol{s}_3$ respectively. Now, given the following two inputs: transformation matrix $\mathbf{A}_e$ from Eq. (3.1) and covariance matrix Σ_{13} from Eq. (3.2), we can compute the desired correlation coefficient ρ_{23} as follows [90]:

$$\rho_{23} = \frac{\boldsymbol{a}_1^\top \Sigma_{13} \boldsymbol{a}_2}{\sqrt{\boldsymbol{a}_1^\top \Sigma_{13} \boldsymbol{a}_1 \cdot \boldsymbol{a}_2^\top \Sigma_{13} \boldsymbol{a}_2}}, \tag{3.3}$$

where $\boldsymbol{a}_1 = (a_{11}, a_{21})$ and $\boldsymbol{a}_2 = (a_{12}, a_{22})$.

It is important to observe the following two advantages regarding the computation of ρ_{23} using Eq. (3.3): first, the computation for ρ_{23} is significantly more efficient as compared to its computation using the original time series $\boldsymbol{s}_2$ and $\boldsymbol{s}_3$ [90]; second, since we do not need the original time series $\boldsymbol{s}_2$ and $\boldsymbol{s}_3$, we require significantly lower memory for computing ρ_{23}. In Section 3.6, we experimentally demonstrate that these advantages manifest a many-fold increase in performance.

Similarly, many other measures of correlation and similarity, beyond the commonly used Pearson's correlation coefficient, can be computed using affine relationships. Thus, by utilizing affine relationships, our approach provides an elegant solution for computing a wide range of statistical measures. As a consequence, our proposal to use affine relationships not only bears the potential of increasing the efficiency of computing the correlation coefficient, but, at the same time, of many other statistical measures.

Measuring quality of affine relationships.

Affine relationships are unlikely to occur over longer real-world time series, however, such relationships may hold approximately, when time series are strongly correlated. For illustration, let us come back to the three stocks from our introductory example. We can compute an approximate affine relationship $\mathbf{A}_e = \left(\begin{smallmatrix} 0.75 & -0.3 \\ 0 & 1 \end{smallmatrix}\right)$ and $\boldsymbol{b}_e = \left(\begin{smallmatrix} 1.6 \\ 0 \end{smallmatrix}\right)$. This relationship is highly accurate between time 150 and 200, but produces errors between time 0 and 50. Therefore, for characterizing such approximation errors we propose a distance metric, *Least Significant Frobenius Distance (LSFD)*, which can take as input, values from stocks $\boldsymbol{s}_1$, $\boldsymbol{s}_2$, and $\boldsymbol{s}_3$ in a specific time window and could quantitatively judge the quality of affine relationships. Additionally, we also propose the AFCLST clustering algorithm that uses the LSFD metric for clustering the time series, such that good-quality affine relationships could be found between cluster members.

We have found that although, in practice, it is almost impossible to find an exact affine relationship between time series, interestingly, a large number of high-quality approximate affine relationships exist in real datasets over longer time intervals. Thus, queries could leverage from such relationships for computing many statistical measures on-the-fly, while bounding the approximation error in the computation of these statistical measures.

Indexing affine relationships.
Consider a slightly modified version of Problem 3.1, where a trader is interested to find all pairs of stock that have the correlation coefficient greater than τ. One way of evaluating this query is to compute – either from scratch or using affine relationships – the correlation coefficient for all the $\frac{n(n-1)}{2}$ pairs, and then return the pairs having correlation coefficient greater than τ. This approach, again, is not scalable for increasing value of n.

A way of circumventing the computation of all the pairwise correlation coefficients is to index the affine relationships. We call this index the SCAPE index. Prior to indexing, the SCAPE index establishes a way of ordering affine relationships. Such an ordering eliminates unnecessary computation and directly gives us the pairs having correlation coefficient greater than τ. Notably, the ordering established by the SCAPE index is agnostic to the underlying statistical measure. As a result, the SCAPE index can be used for simultaneously indexing all the statistical measures.

3.1.1 Chapter Organization

We begin by presenting the details of the AFFINITY framework in Section 3.2. In Section 3.3, we propose the LSFD metric and the AFCLST clustering algorithm for finding high-quality affine relationships in time series data. In Section 3.4, we introduce the SYMEX algorithm for generating high-quality affine relationships, while, in Section 3.5, we propose the SCAPE index for indexing affine relationships. Lastly, comprehensive experimental evaluations are presented in Section 3.6, followed by the review of related studies in Section 3.7.

Table 3.1: Summary of notations.

Symbol	Description
$\mathbf{A}, \ldots$	Matrices (uppercase boldface)
a_{ij}	Entry at row i and column j of matrix $\mathbf{A}$
$\boldsymbol{x}$ or $\boldsymbol{x}_1$	Column vectors (lowercase boldface)
x_i or x_{i1}	element i of a vector $\boldsymbol{x}$ or $\boldsymbol{x}_1$ respectively
$\mathbf{S}$	Data matrix of size $m \times n$
$\mathsf{L}(\mathbf{S}), \mathsf{C}(\mathbf{S}), \mathsf{D}(\mathbf{S})$	Location, dispersion, and derived measures
e, p	Sequence pair and pivot pair
$\mathbf{S}_e$, $\mathbf{O}_p$	Sequence pair matrix and pivot pair matrix
$\mathbb{R}^n$	Set of n-dimensional real column vectors
$\mathbb{R}^{m \times n}$	Set of m-by-n real matrices
$[\boldsymbol{x}_1, \ldots, \boldsymbol{x}_w]$	Column-wise concatenation of w vectors

3.2 Foundation

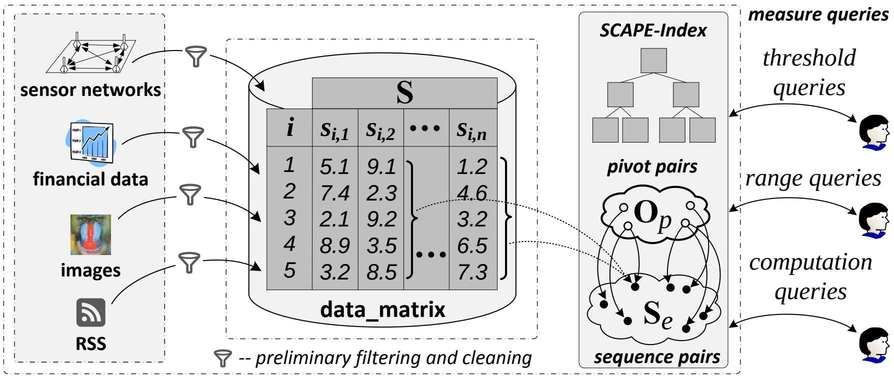
In this section we define the basic concepts and establish the notation used in the rest of the chapter. A summary of the frequently used notations is presented in Table 3.1. Then, we define the queries that are processed by the AFFINITY framework. Most importantly, we discuss the notion of affine transformations and examine their properties. Affine relationships are, in fact, enhanced affine transformations designed for facilitating efficient computation and querying of several statistical measures.

Framework Overview.
Figure 3.2 shows the architecture of the AFFINITY framework. It consists of various time series, like, financial market data, RSS news feeds, sensor network data, etc., that are being stored using a DBMS. AFFINITY consists of two key components: the affine relationships and the SCAPE index structure. The affine relationships are inferred using the data_matrix table, and are indexed for processing statistical queries using the SCAPE index.

Similar to the notation introduced in Section 1.1, let us assume that the AFFINITY framework has n time series and m values per time series, which are stored in the data_matrix table. We compose a matrix consisting of m rows by concatenating the n column vectors as $\mathbf{S} = [\boldsymbol{s}_1, \boldsymbol{s}_2, \ldots, \boldsymbol{s}_n] \in \mathbb{R}^{m \times n}$. We refer to matrix $\mathbf{S}$ as the *data matrix*.

3.2.1 Statistical Measures

In this chapter, we consider three popular classes of statistical measures. The first type of measures are the *location measures* or L-measures that define the central tendency of data (e.g., mean, median, etc.). The second type of measures characterize the joint or pairwise variability in the data and are called *dispersion measures* or C-measures (e.g., covariance, dot product, etc.). The third type are the *derived measures* or D-measures that are



i	$s_{i,1}$	$s_{i,2}$	•••	$s_{i,n}$
1	5.1	9.1		1.2
2	7.4	2.3		4.6
3	2.1	9.2		3.2
4	8.9	3.5	•••	6.5
5	3.2	8.5		7.3

Figure 3.2: Architecture of the AFFINITY framework.

derived by normalizing a dispersion measure, for example, the correlation coefficient is derived by normalizing the covariance.

Often the statistical measures considered in this chapter are required to be computed on pairs of time series. A good example are the covariance and the correlation coefficient. Thus, for conveniently identifying the time series in such scenarios, we define the following two sets. Let $\mathcal{I} = \{u | 1 \leq u \leq n\}$ be the set containing series identifiers $(1, 2, \ldots, n)$ that identify the time series $\boldsymbol{s}_1, \boldsymbol{s}_2, \ldots, \boldsymbol{s}_n$ respectively. We refer to $\mathcal{I}$ as the *series identifier set* and each of its elements as the *series identifier*. Similarly, let $\mathcal{P} = \{(u, v) | u < v \text{ and } (u, v) \in \mathcal{I} \times \mathcal{I}\}$ be the set containing unique pairs of series identifiers. We refer to $\mathcal{P}$ as the *sequence pair set* and each of its elements as the *sequence pair*.

A sequence pair is used for uniquely identifying a pair of time series in the data matrix $\mathbf{S}$. Furthermore, the matrix that is formed by concatenating the time series defined by the sequence pair $e = (u, v) \in \mathcal{P}$ is known as the *sequence pair matrix* and is denoted as $\mathbf{S}_e = [\boldsymbol{s}_u, \boldsymbol{s}_v]$, $\mathbf{S}_e \in \mathbb{R}^{m \times 2}$.

We denote the L-, C-, and D-measures of the matrix $\mathbf{S}$ as $\mathsf{L}(\mathbf{S})$, $\mathsf{C}(\mathbf{S})$ and $\mathsf{D}(\mathbf{S})$ respectively. Here, $\mathsf{L}(\mathbf{S})$ is a vector of size n, and $\mathsf{C}(\mathbf{S})$ and $\mathsf{D}(\mathbf{S})$ are matrices of size $n \times n$. In the matrices $\mathsf{C}(\mathbf{S})$ and $\mathsf{D}(\mathbf{S})$, the entry found at row u and column v is respectively the dispersion and the derived measure between the time series u and v of the matrix $\mathbf{S}$. The entry found at position (u, v) of $\mathsf{C}(\mathbf{S})$ and $\mathsf{D}(\mathbf{S})$ is denoted as $\mathsf{C}_{uv}(\mathbf{S})$ and $\mathsf{D}_{uv}(\mathbf{S})$ respectively. The C- and D-measures are symmetric, that is $\mathsf{C}_{uv}(\mathbf{S}) = \mathsf{C}_{vu}(\mathbf{S})$ and $\mathsf{D}_{uv}(\mathbf{S}) = \mathsf{D}_{vu}(\mathbf{S})$. Secondly, the entry at the position $e = (u, v)$ of the matrix $\mathsf{C}(\mathbf{S})$ denoted as $\mathsf{C}_e(\mathbf{S})$ is equal to $\mathsf{C}_{12}(\mathbf{S}_e)$, which is the entry at position (1,2) of the matrix $\mathsf{C}(\mathbf{S}_e)$. In short, $\mathsf{C}_e(\mathbf{S}) = \mathsf{C}_{12}(\mathbf{S}_e)$ and $\mathsf{D}_e(\mathbf{S}) = \mathsf{D}_{12}(\mathbf{S}_e)$.

In this chapter, we consider three L-measures: mean, mode, and median. In addition, we consider two C-measures: the covariance matrix and the dot product matrix, which are of size n-by-n and are denoted as $\mathit{\Sigma}(\mathbf{S})$ and $\mathit{\Pi}(\mathbf{S})$. We also consider one D-measure, namely, the correlation coefficient matrix denoted as $\rho(\mathbf{S})$. In all these notations subscripts are used to denote specific entries, for example $\mathit{\Pi}_{uv}(\mathbf{S})$ denotes the dot product between time series u and v and $\mathsf{L}_u(\mathbf{S})$ denotes a location measure of the time series u.

We note that all the proposed approaches are also applicable to a large number of other derived measures that are derived by normalizing the dot product; examples of such measures are Jaccard coefficient, Dice coefficient, cosine similarity, harmonic mean, etc.

3.2.2 Query Types

The AFFINITY framework considers three important and frequently-used statistical queries that are posed on time series data. Since our approach supports many statistical measures simultaneously, we generalize the queries by making them independent of the statistical measures. The first query computes a given statistical measure over a requested set of time series, and we define it as follows:

QUERY 3.1: **Measure computation (MEC) query.** Given a set of series identifiers $\psi \subseteq \mathcal{I}$ and a statistical measure (L, C, or D) the *measure computation query* returns the value of the given statistical measure for the time series ψ.

For the C- and D-measures, the response is a matrix of size $|\psi|$-by-$|\psi|$, and for L-measures the response is a vector of size $|\psi|$. For example, the measure computation query could request the mean or the covariance matrix for a subset of the series identifiers ψ.

The second query returns all the series identifiers (sequence pairs) where the location measure (dispersion or derived measure) is greater or lesser than a user-defined threshold.

QUERY 3.2: **Measure threshold (MET) query.** Given a statistical measure L (C or D) and a user-defined threshold τ. The *measure threshold query* returns the set $\mathcal{A}_T$ consisting of the series identifiers u (sequence pairs e) for which the given statistical measure $\mathsf{L}_u(\mathbf{S})$ ($\mathsf{C}_e(\mathbf{S})$ or $\mathsf{D}_e(\mathbf{S})$) is greater or lesser than the threshold τ.

The third query is a range query adaptation of Query 3.2. We define it as follows:

QUERY 3.3: **Measure range (MER) query.** Given a statistical measure L (C or D) and the user-defined lower and upper bounds τ_l and τ_u respectively. The *measure range query* returns the set $\mathcal{A}_R$ consisting of the series identifiers u (sequence pairs e) for which the given statistical measure $\mathsf{L}_u(\mathbf{S})$ ($\mathsf{C}_e(\mathbf{S})$ or $\mathsf{D}_e(\mathbf{S})$) is in between the lower bound τ_l and upper bound τ_u.

An example of the above query could be, return all sequence pairs for which the covariance is in between τ_l and τ_u.

3.2.3 Affine Transformations

Consider any two matrices $\mathbf{X} = [\boldsymbol{x}_1, \boldsymbol{x}_2]$ and $\mathbf{Y} = [\boldsymbol{y}_1, \boldsymbol{y}_2]$, where $\boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{y}_1, \boldsymbol{y}_2$ are column vectors of size m, thus $\mathbf{X}$ and $\mathbf{Y}$ are of size m-by-2. Then, an affine transformation between $\mathbf{X}$ and $\mathbf{Y}$ is defined as:

$$\mathbf{Y} \triangleq \mathbf{X}\mathbf{A} + \mathbb{1}_m \boldsymbol{b}^\top, \tag{3.4}$$

where $\mathbf{A} \in \mathbb{R}^{2\times 2}$ is non-singular, $\boldsymbol{b} \in \mathbb{R}^2$, and $\mathbb{1}_m = (1, 1, \ldots, 1)^\top \in \mathbb{R}^m$ (refer Figure 3.3). We denote the above affine transformation as $(\mathbf{A}, \boldsymbol{b})$. In addition, we denote the first and second column of $\mathbf{A}$ as $\boldsymbol{a}_1$ and $\boldsymbol{a}_2$ respectively. We refer to $\mathbf{X}$ as the *source pair matrix* and $\mathbf{Y}$ as the *target pair matrix*. The difference between an affine transformation and a linear transformation is that an affine transformation is a combination of a linear transformation ($\mathbf{A}$) and a translation ($\boldsymbol{b}$). Therefore, an affine transformation can be considered as a generic form of a linear transformation.

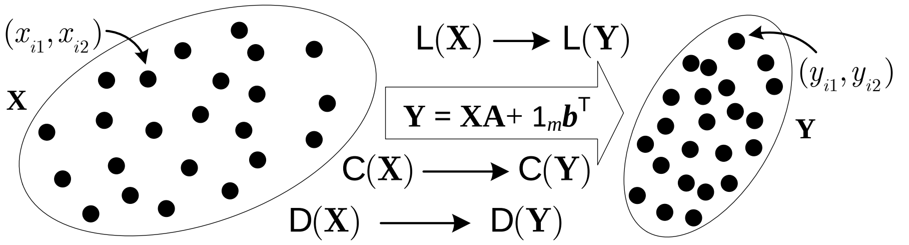

(x_{i1}, x_{i2})
$\mathbf{X}$
$\mathsf{L}(\mathbf{X}) \longrightarrow \mathsf{L}(\mathbf{Y})$
$\mathbf{Y} = \mathbf{X}\mathbf{A} + 1_m \boldsymbol{b}^\top$
$\mathsf{C}(\mathbf{X}) \longrightarrow \mathsf{C}(\mathbf{Y})$
$\mathsf{D}(\mathbf{X}) \longrightarrow \mathsf{D}(\mathbf{Y})$
(y_{i1}, y_{i2})
$\mathbf{Y}$

Figure 3.3: Illustration of an affine transformation.

Interestingly, all the statistical measures that we consider are well-behaved under the action of an affine transformation [90]. Concretely, given the location measure $\mathsf{L}(\mathbf{X})$ of the source pair matrix $\mathbf{X}$, $\mathsf{L}(\mathbf{Y})$ can be computed as:

$$\mathsf{L}(\mathbf{Y})^\top = \mathsf{L}(\mathbf{X})^\top \mathbf{A} + \boldsymbol{b}^\top. \tag{3.5}$$

Similarly, the covariance and the dot product are also well-behaved under the action of an affine transformation. Given the covariance matrix $\varSigma(\mathbf{X})$, $\varSigma(\mathbf{Y})$ can be computed as follows:

$$\varSigma(\mathbf{Y}) = \mathbf{A}^\top \varSigma(\mathbf{X}) \mathbf{A}, \qquad \varSigma_{12}(\mathbf{Y}) = \boldsymbol{a}_1^\top \varSigma(\mathbf{X}) \boldsymbol{a}_2. \tag{3.6}$$

The dot product is well-behaved under the action of an affine transformation as follows [90]:

$$\varPi_{12}(\mathbf{Y}) = \boldsymbol{a}_1^\top \cdot \varPi(\mathbf{X}) \cdot \boldsymbol{a}_2 + \boldsymbol{b}^\top \mathbf{A}^\top \begin{pmatrix} \mathsf{h}_1(\mathbf{X}) \\ \mathsf{h}_2(\mathbf{X}) \end{pmatrix}, \tag{3.7}$$

where $\mathsf{h}_1(\mathbf{X}) = \sum_{i=1}^m x_{i1}$, $\mathsf{h}_2(\mathbf{X}) = \sum_{i=1}^m x_{i2}$.

Additionally, the D-measures are derived by normalizing one of the C-measures. The correlation coefficient is derived by normalizing the covariance as follows:

$$\rho_{12}(\mathbf{Y}) = \frac{\varSigma_{12}(\mathbf{Y})}{\varphi_{12}}, \quad \rho_{12}(\mathbf{Y}) = \frac{\boldsymbol{a}_1^\top \cdot \varSigma_{12}(\mathbf{X}) \cdot \boldsymbol{a}_2}{\varphi_{12}}, \tag{3.8}$$

where φ_{12} is the normalizer and is equal to $\sqrt{\varSigma(\boldsymbol{y}_1)\varSigma(\boldsymbol{y}_2)}$. Observe that the normalizer is separable: $\varSigma(\boldsymbol{y}_1)$ and $\varSigma(\boldsymbol{y}_2)$ can be separately computed. Thus, we simply compute and store $\varSigma(\boldsymbol{y}_1)$ and $\varSigma(\boldsymbol{y}_2)$ separately and then combine them to form φ_{12} as required. We denote the normalizer of the sequence pair e as φ_e .

3.3 Affine Clustering

Consider the problem of computing a statistical measure, say covariance, for all the sequence pairs. The naive approach of solving this problem is to compute the covariance for each pair of sequences from scratch. However, computing covariances from scratch is inefficient because it requires scanning of the sequence pair matrices $\mathbf{S}_e$, for all the sequence pairs e. Since the number of sequence pairs are $\mathcal{O}(n^2)$, where n is the number of time series n, it leads to an overall inefficient operation.

We reduce the $\mathcal{O}(n^2)$ complexity by selecting a small (nearly linear) number of time series pairs, which are called the *pivot pairs*, and the m-by-2 matrices formed by them are called the *pivot pair matrices*; we will shortly describe the selection procedure for the pivot pairs. Then, we compute the covariance for all the pivot pairs and determine the affine transformations between each sequence pair and one of the pivot pairs. Next, with the help of Eq. (3.6) and the affine transformations, we approximate the covariance for all the sequence pairs from the covariance of the pivot pairs. Similarly, other measures can also be only computed for the pivot pairs; and then approximated for the sequence pairs. Note that the affine transformations need to be computed only once.

Next, we describe the selection procedure for the pivot pairs. It should satisfy two requirements: (1) the number of selected pivot pairs should be small, and (2) the affine transformations, when used for approximating a statistical measure, should produce low error. In this section we propose techniques for meeting both these requirements.

3.3.1 Computing the Dot Product

For the dot product, as a special case, we can show that the approximation error can be completely eliminated by having a common time series between the source and target pair matrices. Let us assume that the affine transformation $(\mathbf{A}, \boldsymbol{b})$ is computed using the least-squares method, and it transforms $\mathbf{X}$ to $\mathbf{Y}'$, instead of $\mathbf{Y}$, where $\mathbf{Y}' = [\boldsymbol{y}'_1, \boldsymbol{y}'_2]$. Then, for accurately computing the dot product $\boldsymbol{y}_2^\top \boldsymbol{y}_1$ using affine transformations, we observe that it is sufficient to have one common time series between $\mathbf{X}$ and $\mathbf{Y}$, because of the following lemma:

LEMMA 3.1: The dot product $\boldsymbol{y}_2^\top \boldsymbol{y}_1$ is preserved under the action of an affine transformation $(\mathbf{A}, \boldsymbol{b})$ that is computed using the least-squares method, if $\boldsymbol{y}_1$ is transformed with zero error.

PROOF. Let the hyperplane spanned by vectors $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ be denoted as $\mathcal{H}$. Since $\boldsymbol{y}'_2$ is the least-squares approximation of $\boldsymbol{y}_2$, $\boldsymbol{y}_2 = \boldsymbol{y}'_2 + \epsilon_p$, where ϵ_p is perpendicular to $\mathcal{H}$. Then $\boldsymbol{y}_2^\top \boldsymbol{y}_1 = \boldsymbol{y}_2'^\top \boldsymbol{y}_1 + \epsilon_p^\top \boldsymbol{y}_1$. Since $\boldsymbol{y}_1$ is part of the hyperplane $\mathcal{H}$, $\epsilon_p^\top \boldsymbol{y}_1 = 0$. Hence, $\boldsymbol{y}_2^\top \boldsymbol{y}_1 = \boldsymbol{y}_2'^\top \boldsymbol{y}_1$ □

Obviously, Lemma 3.1 holds even if we replace $\boldsymbol{y}_1$ by $\boldsymbol{y}_2$ and $\boldsymbol{y}'_2$ by $\boldsymbol{y}'_1$. A straightforward way of guaranteeing the transformation of $\boldsymbol{y}_1$ with zero error is to have $\boldsymbol{y}_1$ common to both the source pair and target pair matrices. In this case, we can guarantee that the dot product $\boldsymbol{y}_2^\top \boldsymbol{y}_1$ is accurately computed if the affine transformations are computed using the least-squares method. In addition, as we shall show in Section 3.4, having a common time series reduces the number of pivot pairs, which are generated using the SYMEX algorithm.

3.3.2 Computing Other Measures

For other dispersion and derived measures the exact computation using affine transformations is in general not possible. Therefore, we propose a distance measure for measuring the error in affine transformations, and then a clustering algorithm that helps us identify high-quality affine relationships minimizing this error.

The LSFD Metric.

The *Least Significant Frobenius Distance* (LSFD) metric, when minimized using the clustering algorithm, results in high-quality (i.e., low error) affine transformations between the members of a given cluster. A small LSFD between the source pair matrix $\mathbf{X}$ and the target pair matrix $\mathbf{Y}$ indicates that $\mathbf{X}$ is almost perfectly transformable into $\mathbf{Y}$. The LSFD metric is defined as follows:

DEFINITION 3.1: **LSFD metric.** Suppose $\hat{\mathbf{X}}$ and $\hat{\mathbf{Y}}$ are the zero-mean counterparts of the matrices $\mathbf{X}$ and $\mathbf{Y}$ respectively. Then the Least Significant Frobenius Distance (LSFD) metric is defined as:

$$\mathsf{F}(\mathbf{X}, \mathbf{Y})^2 \triangleq \lambda_3^2 + \lambda_4^2, \tag{3.9}$$

where λ_3 and λ_4 are the third and fourth singular values of the matrix $[\hat{\mathbf{X}}, \hat{\mathbf{Y}}]$, which is a matrix obtained by column-wise concatenation of $\hat{\mathbf{X}}$ and $\hat{\mathbf{Y}}$.

The number of non-zero singular values of a matrix is equal to the number of linearly independent vectors in that matrix. Definition 3.1 assumes that the vectors in $\hat{\mathbf{X}}$ are linearly independent; therefore, if the third and the fourth singular values of the matrix $[\hat{\mathbf{X}}, \hat{\mathbf{Y}}]$ are zero, then it signifies that vectors $\boldsymbol{y}_1$ and $\boldsymbol{y}_2$ are linearly dependent and can be expressed as linear combinations of vectors $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$. Thus, an exact affine transformation between $\mathbf{X}$ and $\mathbf{Y}$ can be computed. Intuitively, the magnitude of the third and the fourth singular value of the matrix $[\hat{\mathbf{X}}, \hat{\mathbf{Y}}]$ quantifies the effort required for making $\boldsymbol{y}_1$ or $\boldsymbol{y}_2$ linearly dependent on $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$.

The LSFD is a metric, and therefore obeys the triangular inequality. Since LSFD is a metric, it can be used as a distance metric for affine clustering. A formal proof of the triangular inequality is presented in the following theorem:

THEOREM 3.1: $\mathsf{F}(\mathbf{X}, \mathbf{Y})$ is a metric; thus $\mathsf{F}(\mathbf{X}, \mathbf{Y})$ obeys the following properties:

- **(a)** *Non-negativity:* $\mathsf{F}(\mathbf{X}, \mathbf{Y})$ is real-valued, finite, and non-negative,
- **(b)** *Positive-definiteness:* $\mathsf{F}(\mathbf{X}, \mathbf{Y}) = 0$ iff there exists an exact affine transformation between $\mathbf{X}$ and $\mathbf{Y}$,
- **(c)** *Symmetry:* $\mathsf{F}(\mathbf{X}, \mathbf{Y}) = \mathsf{F}(\mathbf{Y}, \mathbf{X})$,

(d) *Triangular Inequality:* $\mathsf{F}(\mathbf{X},\mathbf{Y}) \leq \mathsf{F}(\mathbf{X},\mathbf{Z}) + \mathsf{F}(\mathbf{Z},\mathbf{Y})$.

PROOF. **(a)** $\mathsf{F}(\mathbf{X},\mathbf{Y})$ is non-negative since it defined as the sum of squares of two real numbers λ_3 and λ_4. Moreover, since λ_3 and λ_4 are finite and real-valued $\mathsf{F}(\mathbf{X},\mathbf{Y})$ exhibits similar characteristics.

(b) $\Rightarrow$: Let us consider the matrix $\mathbf{I}_{\hat{X}\hat{Y}} = [\hat{\mathbf{X}}, \hat{\mathbf{Y}}]$. $\mathsf{F}(\mathbf{X},\mathbf{Y}) = 0$ implies that λ_3 and λ_4 of the matrix $\mathbf{I}_{\hat{X}\hat{Y}}$ are zero. The number of non-zero singular values of $\mathbf{I}_{\hat{X}\hat{Y}}$ is equal to the rank of $\mathbf{I}_{\hat{X}\hat{Y}}$ denoted by $\text{rank}(\mathbf{I}_{\hat{X}\hat{Y}})$ [52]. If $\lambda_3 = \lambda_4 = 0$ then the rank of the matrix $\text{rank}(\mathbf{I}_{\hat{X}\hat{Y}}) \leq 2$. Therefore, there are maximum two linearly independent columns in the column space of $\mathbf{I}_{\hat{X}\hat{Y}}$. Thus, $4 - \text{rank}(\mathbf{I}_{\hat{X}\hat{Y}})$ columns can be expressed in terms of the $\text{rank}(\mathbf{I}_{\hat{X}\hat{Y}})$ columns or an affine transofrmation exists between $\mathbf{X}$ and $\mathbf{Y}$.

$\Leftarrow$: The argument is similar to $\Rightarrow$, here we use the fact that the existence of an affine transformation indicates that there are at most two linearly independent columns in $\mathbf{I}_{\hat{X}\hat{Y}}$ and therefore $\lambda_3 = \lambda_4 = 0$.

(c) The interchange of columns does not affect the singular values of a matrix [52], hence $\mathbf{F}(\mathbf{X},\mathbf{Y}) = \mathsf{F}(\mathbf{Y},\mathbf{X})$.

(d) Let us consider three matrices $\mathbf{I}_{\hat{X}\hat{Y}} = [\hat{\mathbf{X}}, \hat{\mathbf{Y}}]$, $\mathbf{I}_{\hat{X}\hat{Z}} = [\hat{\mathbf{X}}, \hat{\mathbf{Z}}]$, and $\mathbf{I}_{\hat{Z}\hat{Y}} = [\hat{\mathbf{Z}}, \hat{\mathbf{Y}}]$. Let $\tilde{\mathbf{I}}_{\hat{X}\hat{Y}}$ be the rank two approximation of the matrix $\mathbf{I}_{\hat{X}\hat{Y}}$. The Frobenius norm of $\|\mathbf{I}_{\hat{X}\hat{Y}} - \tilde{\mathbf{I}}_{\hat{X}\hat{Y}}\|_F$ is $\sqrt{\lambda_3^2 + \lambda_4^2}$. Similarly, let $\tilde{\mathbf{I}}_{\hat{X}\hat{Z}}$ and $\tilde{\mathbf{I}}_{\hat{Z}\hat{Y}}$ be the rank two approximations of the matrices $\mathbf{I}_{\hat{X}\hat{Z}}$ and $\mathbf{I}_{\hat{Z}\hat{Y}}$ respectively. Then,

$$\mathbf{I}_{\hat{X}\hat{Y}} = \mathbf{I}_{\hat{X}\hat{Z}} + \mathbf{I}_{\hat{Z}\hat{Y}} + [-\hat{\mathbf{Z}}, -\hat{\mathbf{Z}}]. \tag{3.10}$$

Let $\mathbf{B} = [-\hat{\mathbf{Z}}, -\hat{\mathbf{Z}}] + \tilde{\mathbf{I}}_{\hat{X}\hat{Z}} + \tilde{\mathbf{I}}_{\hat{Z}\hat{Y}}$. From Eq. (3.10),

$$\|\mathbf{I}_{\hat{X}\hat{Y}} - \mathbf{B}\|_F \leq \|\mathbf{I}_{\hat{X}\hat{Z}} - \tilde{\mathbf{I}}_{\hat{X}\hat{Z}}\|_F + \|\mathbf{I}_{\hat{Z}\hat{Y}} - \tilde{\mathbf{I}}_{\hat{Z}\hat{Y}}\|_F.$$

Using the Eckart-Young low-rank matrix approximation theorem [52] and the definition of LSFD in Definition 3.1,

$$\begin{gathered}\|\mathbf{I}_{\hat{X}\hat{Y}} - \tilde{\mathbf{I}}_{\hat{X}\hat{Y}}\|_F \leq \|\mathbf{I}_{\hat{X}\hat{Z}} - \tilde{\mathbf{I}}_{\hat{X}\hat{Z}}\|_F + \|\mathbf{I}_{\hat{Z}\hat{Y}} - \tilde{\mathbf{I}}_{\hat{Z}\hat{Y}}\|_F, \\ \mathsf{F}(\mathbf{X},\mathbf{Y}) \leq \mathsf{F}(\mathbf{X},\mathbf{Z}) + \mathsf{F}(\mathbf{Z},\mathbf{Y}). \end{gathered} \tag{3.11}$$

□

3.3.3 The AFCLST Clustering Algorithm

The affine clustering algorithm clusters the time series in the data matrix $\mathbf{S}$ into k clusters, such that it becomes easier to identify a high-quality affine transformation between a sequence pair and a pivot pair. We have one common time series between the sequence pair matrix and the pivot matrix for computing the dot product accurately. As a result the common time series is transformed with zero LSFD error. Next, for the other (different) time series in the sequence pair matrix, the affine clustering algorithm finds the closest match, such that the LSFD between the sequence pair matrix and the pivot pair matrix becomes as low as possible.

The closest match for the different time series is its cluster center, which is returned by the affine clustering algorithm. We show that by following this procedure for constructing the pivot pair matrix, the LSFD between the pivot pair matrix and the sequence pair matrix is minimized, resulting in high-quality affine transformations. Thus, the pivot pair matrix – like the source pair matrix $\mathbf{X}$ – can be utilized for accurately computing the statistical measures over the sequence pair matrix.

The affine clustering algorithm clusters the time series in $\mathbf{S}$ into k clusters (refer Algorithm 3.1). It starts by initializing the cluster centers $\boldsymbol{r}_\ell$, where $\ell = (1, \ldots, k)$ (Lines 2 and 3). In the assignment step, the AFCLST algorithm computes the orthogonal projection of each time series $\boldsymbol{s}_v$, where $1 \leq v \leq n$, onto the cluster centers $\boldsymbol{r}_\ell$, and assigns $\boldsymbol{s}_v$ to the cluster that produces the least projection error $proj_\epsilon$ (Lines 10, 15, and Fig. 3.4(b)). The lesser the orthogonal projection error $proj_\epsilon$, the more accurately a time series can be represented by a linear combination of its cluster center; leading to a lower LSFD between the sequence pair matrix and the pivot matrix.

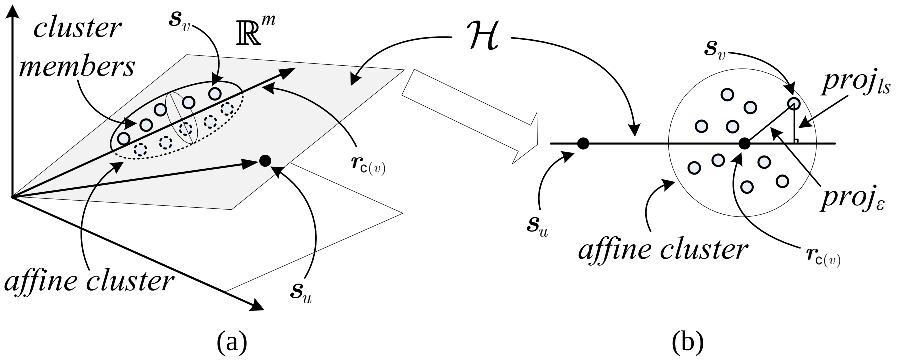


Figure 3.4: (a) the 2-D hyperplane $\mathcal{H}$, and (b) directional view of the hyperplane $\mathcal{H}$.

In the update phase, the cluster centers $\boldsymbol{r}_\ell$ are re-computed. This is done by forming a matrix $\mathbf{R}_\ell$ for each cluster ℓ, by column-wise concatenation of the time series assigned to cluster ℓ. The updated cluster center is equal to the left singular vector associated with the largest singular value of $\mathbf{R}_\ell$. Intuitively, the left singular vector associated with the largest singular value is the one that minimizes the sum of the orthogonal projection errors that are computed between the cluster center of cluster ℓ (i.e., $\boldsymbol{r}_\ell$) and each of its members.

The AFCLST algorithm terminates when the number of cluster membership changes is less than $\min(\delta)$ or the maximum number of iterations $\max(\gamma)$ are completed. The AFCLST algorithm returns two quantities: (a) cluster centers $\boldsymbol{r}_1, \boldsymbol{r}_2, \ldots, \boldsymbol{r}_k$ and (b) a cluster assignment function $\mathsf{c}(v) : v \mapsto \ell$, which returns the cluster identifier ℓ for a given series identifier v.

For a given sequence pair $e = (u, v)$, we now form a pivot pair matrix $[\boldsymbol{s}_u, \boldsymbol{r}_{\mathsf{c}(v)}]$, obtained by concatenating the time series $\boldsymbol{s}_u$ and the cluster center of the time series $\boldsymbol{s}_v$. Furthermore, let $\mathcal{H}$ be the 2-D hyperplane spanned by the vectors $\boldsymbol{s}_u$ and $\boldsymbol{r}_{\mathsf{c}(v)}$ (refer Figure 3.4). Then there exists a high-quality affine transformation between the pivot pair matrix $[\boldsymbol{s}_u, \boldsymbol{r}_{\mathsf{c}(v)}]$ and the sequence pair matrix $\mathbf{S}_e = [\boldsymbol{s}_u, \boldsymbol{s}_v]$. This is true since the projection error $proj_{ls}$, obtained from orthogonally projecting $\boldsymbol{s}_v$ onto $\mathcal{H}$, can only be less than $proj_\epsilon$. From Fig. 3.4(b), $proj_{ls}$ is one side of the right-angled triangle where $proj_\epsilon$ is an hypotenuse. Thus, approximating $\boldsymbol{s}_v$ using $[\boldsymbol{s}_u, \boldsymbol{r}_{\mathsf{c}(v)}]$ further reduces the LSFD.

Now, we present formal, crisp definitions of the pivot pair and the pivot pair matrix, associated to a sequence pair $e = (u, v) \in \mathcal{P}$:

Definition 3.2: **Pivot pair and pivot pair matrix.** The *pivot pair* associated to the sequence pair $e = (u, v)$ is defined as $p = (u, \mathsf{c}(v))$. Moreover, it is a sequence pair where the series identifier v is replaced by its cluster identifier $\mathsf{c}(v)$. The *pivot pair matrix*, denoted as $\mathbf{O}_p$, is the matrix obtained by concatenating the time series $\boldsymbol{s}_u$ with the cluster center $\boldsymbol{r}_{\mathsf{c}(v)}$ as follows:

$$\mathbf{O}_p \triangleq [\boldsymbol{s}_u, \boldsymbol{r}_{\mathsf{c}(v)}]. \tag{3.12}$$

Observe that $(\mathsf{c}(u), v)$ is also considered a pivot pair, but for reasons of brevity we only use Definition 3.2 of a pivot pair. We end this section by defining the most crucial concept proposed in this chapter – *affine relationships*:

Algorithm 3.1 The AFCLST affine clustering algorithm.

```
Input: Data matrix $\mathbf{S}$, maximum iterations $\max(\gamma)$, number of clusters $k$, minimum cluster changes $\min(\delta)$.
Output: Cluster centers $\boldsymbol{r}_\ell$ and cluster assignment function $\mathsf{c}(v)$.
for $\ell = 1$ to $k$ do                                   ▷ Initialization phase
    $\boldsymbol{r}_\ell \leftarrow$ RANDCOL($\mathbf{S}$)        ▷ choose a random column
    $\boldsymbol{r}_\ell \leftarrow \boldsymbol{r}_\ell / \|\boldsymbol{r}_\ell\|$        ▷ normalize
$nChg \leftarrow -1$
for $iter = 1$ to $\max(\gamma)$ do
    $minProj_\epsilon \leftarrow \infty$, $clustID \leftarrow 0$
    for $j = 1$ to $m$ do                                  ▷ Assignment phase
        for $\ell = 1$ to $k$ do
            $proj_{\boldsymbol{r}_\ell} \leftarrow (\boldsymbol{r}_\ell \boldsymbol{r}_\ell^\top)\boldsymbol{s}_j$
            $proj_\epsilon \leftarrow ||proj_{\boldsymbol{r}_\ell} - \boldsymbol{s}_j||$
            if $proj_\epsilon < minProj_\epsilon$ then
                $clustID \leftarrow \ell$
        if $\mathsf{c}(j)$ != $clustID$ then
            $currNChg \leftarrow currNChg + 1$
        $\mathsf{c}(j) \leftarrow clustID$
    if $|nChg - currNChg| \leq \min(\delta)$ then
        break                                              ▷ Converged
    for $\ell = 1$ to $k$ do                               ▷ Update phase
        $\mathbf{R}_\ell \leftarrow \emptyset$
        for $j = 1$ to $m$ do
            if $\mathsf{c}(j) == \ell$ then
                $\mathbf{R}_\ell \leftarrow [\mathbf{R}_\ell, \boldsymbol{s}_j]$
        $\boldsymbol{r}_\ell \leftarrow$ SVDLV($\mathbf{R}_\ell$)          ▷ Largest left singular vector
return $\boldsymbol{r}_\ell, \mathsf{c}(u)$
```

DEFINITION 3.3: **Affine relationship.** An affine relationship characterizes a relationship between the sequence pair e and its pivot pair p. Precisely, it is defined as an affine transformation between the sequence pair matrix $\mathbf{S}_e$ and the pivot pair matrix $\mathbf{O}_p$,

$$\mathbf{S}_e \triangleq \mathbf{O}_p\mathbf{A}_e + \mathbb{1}_m \boldsymbol{b}_e^\top, \tag{3.13}$$

where $\mathbf{A}_e \in \mathbb{R}^{2\times2}$ is non-singular, $\boldsymbol{b}_e \in \mathbb{R}^2$, and $\mathbb{1}_m = (1, 1, \ldots, 1)^\top \in \mathbb{R}^m$. We use $(\mathbf{A}, \boldsymbol{b})_e$ to denote an affine relationship.

To summarize, we use the following procedure for selecting a pivot pair p for a given sequence pair e. First, we keep a common time series, namely $\boldsymbol{s}_u$, between the sequence pair matrix and the pivot pair matrix. Second, the other (uncommon) time series in the pivot pair matrix is the affine cluster center of the time series $\boldsymbol{s}_v$. By this procedure, the pivot pair of the sequence pair $e = (u, v)$ is $p = (u, \mathsf{c}(v))$. In the following section we propose an algorithm that systematically follows this procedure for generating pivot pairs p that correspond to sequence pairs e

3.4 Computing Affine Relationships

In this section we propose an algorithm for generating the pivot pairs p for the given sequence pairs e using the procedure described in Section 3.3. Secondly, we propose a method for efficiently computing the affine relationships between the selected pivot and sequence pairs.

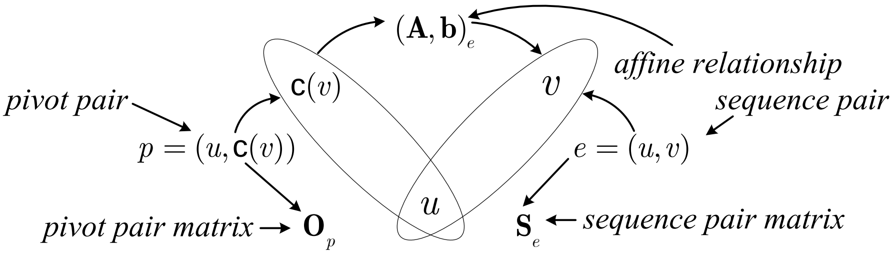

$(\mathbf{A}, \mathbf{b})_e$
affine relationship
sequence pair
pivot pair
$\mathsf{c}(v)$
v
$p = (u, \mathsf{c}(v))$
$e = (u, v)$
u
pivot pair matrix → $\mathbf{O}_p$
$\mathbf{S}_e$ ← sequence pair matrix

Figure 3.5: Procedure for generating the pivot pairs.

The proposed algorithm follows the following steps (refer Figure 3.5): (i) select any sequence pair $e = (u, v) \in \mathcal{P}$, (ii) generate both the possible pivot pairs for e: $(u, \mathsf{c}(v))$ and $(\mathsf{c}(u), v)$, (iii) associate the pivot pair $(u, \mathsf{c}(v))$ to the sequence pair e, and then form a new sequence pair by changing the second component of e to another member of cluster $\mathsf{c}(v)$, (iv) repeat Step (iii) with the new sequence pair, until all the members of the cluster $\mathsf{c}(v)$ have been associated the pivot pair $(u, \mathsf{c}(v))$, (v) use the other pivot pair $(\mathsf{c}(u), v)$ and repeat Step (iii), now changing the first component, and (vi) jump to Step (i) if there are more sequence pairs that have not been associated a pivot pair.

A formal algorithm of the Steps (i)-(vi) is presented in Algorithm 3.2. The only difference is that, instead of selecting any sequence pair, as per Step (i), Algorithm 3.2 selects them systematically. The algorithm starts processing the sequence pair set $\mathcal{P}$

using two fixed sequence pairs: $e_e = (1, n)$ and $e_w = (\frac{n-1}{2}, \frac{n-1}{2} + 1)$. Then, it generates new sequence pairs by alternatively adding $(1, -1)$ and $(-1, 1)$ to e_e and e_w respectively (Line 6 and Line 9). On Line 14, it scans each component of the new sequence pair, until the boundary of the set $\mathcal{P}$ is reached.

During each step of the scan it associates a sequence pair e to a pivot pair p, only if the sequence pair e has not been associated with a pivot pair earlier (Line 20). On Line 21, e and p are used for computing the affine relationship $(\mathbf{A}, \boldsymbol{b})_e$. These affine relationships are stored in the hash map affHash. affHash is returned by the algorithm along with another hash map pivotHash, which stores the pivot pairs generated by the algorithm. The algorithm stops when both the sequence pairs e_e and e_w are equal. Since Algorithm 3.2 systematically selects the sequence pairs, we refer to it as the SYMEX (Systematic Exploration of $\mathcal{P}$) algorithm.

The SYMEX algorithm produces maximum nk number of pivot pairs, where k is the number of affine clusters. But in practice we found that $k << n$, thus the SYMEX algorithm produces pivot pairs nearly linear in the number of time series n. Moreover, the complexity of the SYMEX algorithm is $\mathcal{O}(g)$, where g is the number of affine relationships produced by the algorithm; thus it is linear in the number of affine relationships g. In Section 3.6, we perform experiments for demonstrating the linear scalability of the SYMEX algorithm.

Lastly, we stress the fact that in the SYMEX algorithm it is not necessary to store and track all the affine relationships. We can, if required, prune the unnecessary affine relationships on the basis of domain knowledge, query requirements etc. This, however, is not the main focus of this chapter, and would be considered in subsequent works. On the contrary, here we consider all the affine relationships returned by the SYMEX algorithm, for clearly demonstrating performance and scalability results.

Pseudo-inverse cache.

Notice that, on Line 26, the SYMEX algorithm computes the pseudo-inverse of the matrix $[\mathbf{O}_p, \mathbb{1}_m]$. This is necessary for solving the system of equations for finding $\mathbf{A}$ and $\boldsymbol{b}$ by the least-squares method. Since there are many sequence pairs e associated to a single pivot pair p, the same pseudo-inverse of $[\mathbf{O}_p, \mathbb{1}_m]$ is repeatedly re-computed for each pivot pair.

Thus, we propose another variant of the SYMEX algorithm that caches, instead of re-computing, the pseudo-inverse. We call this variant the SYMEX+ algorithm. The proposed pseudo-inverse cache is populated by inserting the pseudo-inverse of $[\mathbf{O}_p, \mathbb{1}_m]$ with key p, before the calls to the SolveInsert function (Line 15 and Line 18). Then, the pseudo-inverse is only computed if the cache lookup is unsuccessful. As we shall demonstrate in Section 3.6, the SYMEX+ algorithm is a factor of *4 times* faster as compared to the SYMEX algorithm.

Algorithm 3.2 The SYMEX algorithm.

```
Input: Data matrix $\mathbf{S}$, AFCLST algorithm parameters $k, \max(\gamma)$, and $\min(\delta)$.
Output: Hash maps affHash and pivotHash, containing the affine relationships and the pivot
    pairs respectively.
$(\boldsymbol{r}_\ell, \mathsf{c}(u)) \leftarrow \mathrm{AFCLST}(\mathbf{S}, k, \max(\gamma), \min(\delta))$
$e_e \leftarrow (0, n)$, $e_w \leftarrow (\frac{n-1}{2}, \frac{n-1}{2} + 1)$                ▷ sequence pairs
$flip \leftarrow 0$
while $e_e$ != $e_w$ do
    if $flip == 0$ then
        $e_e \leftarrow e_e + (1, -1)$, $flip \leftarrow 1$                             ▷ move towards $e_w$
        CreatePivots($e_e$, affHash)
    else if $flip == 1$ then
        $e_w \leftarrow e_w + (-1, 1)$, $flip \leftarrow 0$                             ▷ move towards $e_e$
        CreatePivots($e_w$, affHash)
return affHash
function CreatePivots($e_z = (u_z, v_z)$, affHash)
    for $v = u_z + 1$ to $n$ do                                                ▷ Scan second component
        $e \leftarrow (u_z, v)$, $p \leftarrow (u_z, \mathsf{c}(v))$
        SolveInsert($\mathbf{O}_p$, $\mathbf{S}_e$, affHash)
    for $u = 0$ to $v_z$ do                                                    ▷ Scan first component
        $e \leftarrow (u, v_z)$, $p \leftarrow (\mathsf{c}(u), v_z)$
        SolveInsert($\mathbf{O}_p$, $\mathbf{S}_e$, affHash)
function SolveInsert($\mathbf{O}_p$, $\mathbf{S}_e$, affHash)
    if affHash.$lookup(e) == \emptyset$ then
        $(\mathbf{A}, \boldsymbol{b}) \leftarrow$ LeastSquares($\mathbf{O}_p$, $\mathbf{S}_e$)
        affHash.$insert(e, (\mathbf{A}, \boldsymbol{b}))$                              ▷ $insert(key, value)$
    if pivotHash.$lookup(p) == \emptyset$ then
        pivotHash.$insert(p, \emptyset)$                                          ▷ null hash values
function LeastSquares($\mathbf{O}_p$, $\mathbf{S}_e$)
    $pinv \leftarrow$ PseudoInv($[\mathbf{O}_p, \mathbb{1}_m]$)                              ▷ Pseudo-inverse
    $(\mathbf{A}, \boldsymbol{b}) \leftarrow pinv \cdot \mathbf{S}_e$
    return $(\mathbf{A}, \boldsymbol{b})$
```

3.4.1 Measure Computation Query

We discuss the processing of Query 3.1, or the MEC query, using affine relationships. Assume that the MEC query has requested to compute the covariance matrix of the series identifiers ψ. Let us denote the sequence pairs formed by the series identifiers ψ as $e_\psi \in \mathcal{P}$.

The first step is the pre-processing step. This step fills the values in the empty hash map pivotHash, which is returned by the SYMEX+ algorithm. For each pivot pair p, contained in the pivotHash hash map, the value is set to the covariance matrix of the pivot pair matrix $\mathbf{O}_p$. Our task is to compute $\Sigma_{e_\psi}(\mathbf{S})$ for each sequence pair $e_\psi \in \mathcal{P}$. For performing this task we search for two things: (a) covariance of the pivot pair p_ψ in the pivotHash hash map, denoted as $\Sigma(\mathbf{O}_{p_\psi})$, and (b) affine relationship $(\mathbf{A}, \boldsymbol{b})_{e_\psi}$ for the sequence pair e_ψ in the affHash hash map. Using these inputs and Eq. (3.6) we compute $\Sigma_{e_\psi}(\mathbf{S}_\psi)$ as:

$$\Sigma_{e_\psi}(\mathbf{S}) = \Sigma_{12}(\mathbf{S}_{e_\psi}) = \boldsymbol{a}_1^\top \Sigma(\mathbf{O}_{p_\psi}) \boldsymbol{a}_2, \tag{3.14}$$

where $\boldsymbol{a}_1 = (a_{11}, a_{21})^\top$ and $\boldsymbol{a}_2 = (a_{12}, a_{22})^\top$ are the first and second columns of the transformation matrix $\mathbf{A}_{e_\psi}$. This procedure is followed for all the other sequence pairs e_ψ.

Similarly, a MEC query requesting computation of a L-measure, dot product, or D-measure can be processed using their corresponding properties in Eqs. (3.5), (3.7) and (3.8) along with the output of the SYMEX+ algorithm. For the D-measures the separable normalizers are computed in the pre-processing step and then utilized for normalization.

Cost Analysis: The total computational cost of the MEC query can be divided into three parts: (a) a one-time cost of order $\mathcal{O}(nk)$ for computing and storing the covariance matrices of all the nk pivot pairs, (b) the average run-time cost of finding an affine relationship from affHash is of order $\mathcal{O}(1)$, and (c) a small cost for computing the requested measure using Eq. (3.6). As it can be seen, the one-time cost $\mathcal{O}(nk)$ of (a) dominates the Big-$\mathcal{O}$ complexity. In contrast, the naive approach always computes all the covariance matrices, which are of order $\mathcal{O}(n^2)$. Moreover, as we shall see in Section 3.6, since $k << n$ in practice this dominating one-time cost becomes nearly linear in the number of time series n, leading to significantly large performance improvements.

Error Measurement: Another important issue is the error measure used for characterizing the approximation error. Suppose $\hat{\Sigma}_e(\mathbf{S})$ and $\Sigma_e(\mathbf{S})$ respectively are the true value (computed from scratch) and the approximated value (computed using affine relationships) of the covariance for the sequence pair e. We, then, compute the normalized values $\hat{\Sigma}^n_e(\mathbf{S})$ and $\Sigma^n_e(\mathbf{S})$, by dividing $\hat{\Sigma}_e(\mathbf{S})$ and $\Sigma_e(\mathbf{S})$ with a normalizer $(\max(\hat{\Sigma}_e(\mathbf{S})) - \min(\hat{\Sigma}_e(\mathbf{S})))$, where the maximum and the minimum are computed over all the sequence pairs in $\mathcal{P}$. Next, we compute the RMSE (root-mean-square error) between the normalized values as follows:

$$\%\ \text{RMSE} = \sqrt{\frac{\sum_{e\in\mathcal{P}} \left(\hat{\Sigma}^n_e(\mathbf{S}) - \Sigma^n_e(\mathbf{S})\right)^2}{|\mathcal{P}|}} \times 100 \tag{3.15}$$

3.5 Indexing Affine Relationships

In this section we propose efficient methods for processing the MET and MER queries described in Section 3.2.2. A straight forward way of processing these queries is to either use the naive approach or the affine relationships approach to first compute the value of the queried statistical measure and then trivially evaluate the MET and MER queries.

A major drawback of this approach is that we have to re-compute the queried statistical measure for every query and for all sequence pairs, which makes this approach inefficient, especially when large number of queries are processed. In contrast, the Scalar Projection or SCAPE index is designed in such a way that: (a) queries over all the statistical measures can be processed without re-computing the measure for every query, and (b) a single index can process queries for all the L-, C-, and D-measures. Furthermore, using the SCAPE index we can improve the efficiency of processing the MET and MER queries by orders of magnitude.

The SCAPE index consists of a sorted container, like a B-tree, for each pivot pair. Each sorted container, associated to a pivot pair, stores the affine relationships assigned to that pivot pair. The key used for sorting is the most crucial and novel component of the SCAPE index. The key chosen for the SCAPE index should be measure-independent, only then we can index all the statistical measures using the same index. Additionally, the key should be such that a query (MET or MER) over any statistical measure could be converted into a query (MET or MER) over the keys stored by the SCAPE index, guaranteeing that the results from the converted and the original query are the same.

For choosing a key with the above properties, the SCAPE index uses an interesting property of the scalar product between two vectors. Let us briefly understand this property through an example. Suppose we have a vector $\boldsymbol{\alpha}$ and vectors $\boldsymbol{\beta}_l$, where l is a positive integer, and our objective is to order the scalar product $\boldsymbol{\alpha}^\top\boldsymbol{\beta}_l \in \mathbb{R}$. Then, the scalar product can be defined as $\boldsymbol{\alpha}^\top\boldsymbol{\beta}_l = \|\boldsymbol{\alpha}\| \cdot \|\boldsymbol{\beta}_l\| \cos(\theta_l)$, where θ_l is the angle between $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}_l$. Notice, $\|\boldsymbol{\alpha}\|$ is common to all the $\|\boldsymbol{\alpha}\| \cdot \|\boldsymbol{\beta}_l\| \cos(\theta_l)$, therefore it is sufficient to use $\|\boldsymbol{\beta}_l\| \cos(\theta_l)$ as a key for ordering the scalar product (refer Figure 3.6). $\|\boldsymbol{\beta}_l\| \cos(\theta_l)$ is known as the *scalar projection* of $\boldsymbol{\beta}_l$ on $\boldsymbol{\alpha}$, and is denoted as ξ_l. The above example encourages us to formulate the following observation:

OBSERVATION 3.1: Given a vector $\boldsymbol{\alpha}$ and vectors $\boldsymbol{\beta}_l$, where l is a positive integer. The scalar projections $\xi_l = \|\boldsymbol{\beta}_l\| \cos(\theta_l)$ can be used as a key for ordering the scalar products $\boldsymbol{\alpha}^\top\boldsymbol{\beta}_l$.

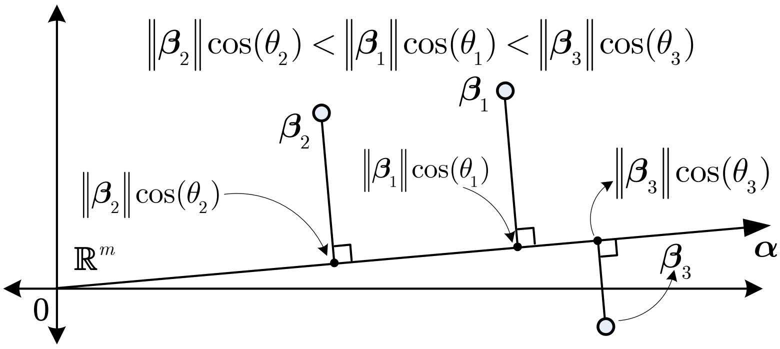

$\|\boldsymbol{\beta}_2\| \cos(\theta_2) < \|\boldsymbol{\beta}_1\| \cos(\theta_1) < \|\boldsymbol{\beta}_3\| \cos(\theta_3)$
$\boldsymbol{\beta}_1$
$\boldsymbol{\beta}_2$
$\|\boldsymbol{\beta}_2\| \cos(\theta_2)$
$\|\boldsymbol{\beta}_1\| \cos(\theta_1)$
$\|\boldsymbol{\beta}_3\| \cos(\theta_3)$
$\mathbb{R}^m$
$\boldsymbol{\alpha}$
$\boldsymbol{\beta}_3$
0

Figure 3.6: Toy example demonstrating Observation 3.1.

3.5.1 Scalar Projection (SCAPE) Index

Now let us discuss the application of Observation 3.1 for indexing affine relationships. Assume that we obtained Q pivot pairs by executing the SYMEX+ algorithm described in Section 3.4. Let us denote them as p_q where $q = (1, 2, \dots Q)$. Also, assume that each pivot pair p_q has D_q sequence pairs associated with it. Let us denote these sequence pairs as e_{qd} where $d = (1, 2, \dots D_q)$. Suppose we are interested in processing the MET and MER queries for the covariance. Recall, given the affine relationship $(\mathbf{A}, \boldsymbol{b})_{e_{qd}}$ for a sequence pair e_{qd} and the covariance matrix of the pivot matrix $\Sigma(\mathbf{O}_{p_q})$, the covariance $\Sigma_{e_{qd}}(\mathbf{S})$ can be estimated as follows:

$$\Sigma_{e_{qd}}(\mathbf{S}) = \boldsymbol{a}_2^\top \Sigma(\mathbf{O}_{p_q})\boldsymbol{a}_1, \tag{3.16}$$

where $\boldsymbol{a}_1$ and $\boldsymbol{a}_2$ are first and second column of the transformation matrix $\mathbf{A}_{e_{qd}}$. Since from Definition 3.2, we have a common time series, namely u, between the sequence pair e_{qd} and the pivot pair p_q, it simplifies the structure of $\boldsymbol{a_1}$ to $(1, 0)^\top$. Thus, Eq. (3.16) becomes:

$$\Sigma_{e_{qd}}(\mathbf{S}) = (a_{12}, a_{22}) \begin{pmatrix} \Sigma_{11}(\mathbf{O}_{p_q}) \\ \Sigma_{21}(\mathbf{O}_{p_q}) \end{pmatrix}. \tag{3.17}$$

We then define $\boldsymbol{\alpha}_q = (\Sigma_{11}(\mathbf{O}_{p_q}), \Sigma_{21}(\mathbf{O}_{p_q}))^\top$, $\boldsymbol{\beta}_{qd} = (a_{12}, a_{22})^\top$ and thus $\Sigma_{e_{qd}}(\mathbf{S}) = \boldsymbol{\alpha}_q^\top \boldsymbol{\beta}_{qd}$. Now, similar to Observation 3.1, for ordering the scalar products $\boldsymbol{\alpha}_q^\top \boldsymbol{\beta}_{qd}$ it is sufficient to order only the scalar projections $\xi_{qd} = \|\boldsymbol{\beta}_{qd}\| \cos(\theta_{qd})$, where θ_{qd} is the angle between $\boldsymbol{\alpha}_q$ and $\boldsymbol{\beta}_{qd}$. Notice that $\boldsymbol{\beta}_{qd}$ is derived only using the affine relationships, and does not change even if $\boldsymbol{\alpha}_q$ changes. Thus, we have essentially decoupled the affine relationship (captured by $\boldsymbol{\beta}_{qd}$) from the statistical measure (captured by $\boldsymbol{\alpha}_q$).

This decoupling allows us to define an $\boldsymbol{\alpha}_q$ for other measures without affecting the ordering of the key ξ_{qd}. Thus, like covariance, we can find an $\boldsymbol{\alpha}_q$ for the other L-measures and the dot product. Table 3.2 summarizes the values of $\boldsymbol{\alpha}_q$ and $\boldsymbol{\beta}_{qd}$ for all the L- and C-measures. In summary, by using the same ordering of ξ_{qd} we can index all the L- and C-measures considered in this chapter.

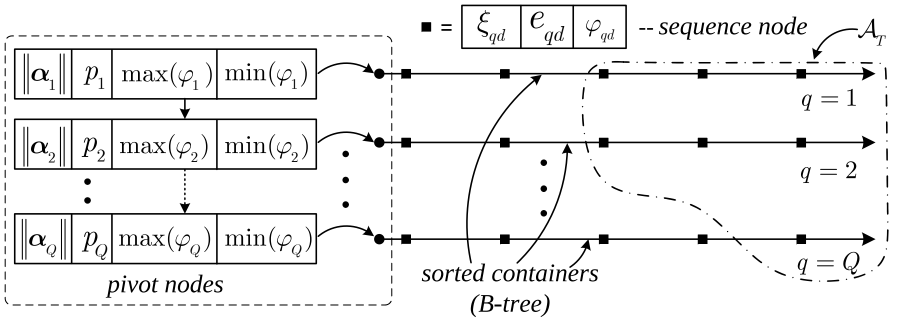

■ = ξ_{qd} e_{qd} φ_{qd} -- sequence node
$\mathcal{A}_T$
$\|\boldsymbol{\alpha}_1\|$ p_1 $\max(\varphi_1)$ $\min(\varphi_1)$
$\|\boldsymbol{\alpha}_2\|$ p_2 $\max(\varphi_2)$ $\min(\varphi_2)$
$\|\boldsymbol{\alpha}_Q\|$ p_Q $\max(\varphi_Q)$ $\min(\varphi_Q)$
pivot nodes
$q = 1$
$q = 2$
$q = Q$
sorted containers (B-tree)

Figure 3.7: Example of the SCAPE index for indexing a C-measure and a D- measure.

Moreover, the SCAPE index contains two types of nodes: (a) *pivot node* that includes the pivot pair p_q and $\|\boldsymbol{\alpha}_q\|$ for all the statistical measures that are indexed by the SCAPE index, and (b) *sequence node* that includes the sequence pair e_{qd} and the scalar projection $\xi_{qd} = \|\boldsymbol{\beta}_{qd}\| \cos(\theta_{qd})$. Furthermore, all the sequence nodes, associated with a pivot node, are stored in a sorted container, like a B-tree. The key for sorting is the scalar projection ξ_{qd}, which is found in each sequence node. In addition, each pivot node also stores the reference to the sorted container that stores its sequence nodes. A schematic depicting the arrangement between the pivot nodes and the sequence nodes is shown in Figure 3.7.

Table 3.2: Choices of $\boldsymbol{\alpha}_q$ and $\boldsymbol{\beta}_{qd}$. The third column refers to the affine relationship $(\mathbf{A}, \boldsymbol{b})$ between p_q and e_{qd}.

	$\boldsymbol{\alpha}_q$	$\boldsymbol{\beta}_{qd}$
Location $\mathsf{L}_2(\mathbf{S}_{e_{qd}})$	$(\mathsf{L}_1(\mathbf{O}_{p_q}), \mathsf{L}_2(\mathbf{O}_{p_q}), 1)^\top$	$(a_{12}, a_{22}, b_2)^\top$
Covariance $\varSigma_{12}(\mathbf{S}_{e_{qd}})$	$(\varSigma_{12}(\mathbf{O}_{p_q}), \varSigma_{22}(\mathbf{O}_{p_q}), 0)^\top$	$(a_{12}, a_{22}, b_2)^\top$
Dot product $\varPi_{12}(\mathbf{S}_{e_{qd}})$	$(\varPi_{12}(\mathbf{O}_{p_q}), \varPi_{11}(\mathbf{O}_{p_q}), \mathsf{h}_1(\mathbf{O}_{p_q}))$	$(a_{12}, a_{22}, b_2)^\top$

Thus, in short, using the SCAPE index we have essentially indexed all the L- and C-measures at once.

Indexing D-Measures: For indexing a D-measure, we should additionally store the following two values with the existing SCAPE index structure. First, in each sequence node, the normalizer φ_{qd} of the indexed D measure, for e.g., $\sqrt{\varSigma(\boldsymbol{s}_u)\varSigma(\boldsymbol{s}_v)}$ for the correlation coefficient. Second, in each pivot node, the maximum and the minimum value of the normalizer, $\max(\varphi_q)$ and $\min(\varphi_q)$, found in the B-tree associated with the pivot pair p_q.

In Section 3.5.3, we show that the above two quantities are sufficient to prune the SCAPE index and efficiently process the MET and MER queries on the D-measures. Similarly, other D-measures, which are not included in this chapter, can also be indexed with the SCAPE index.

3.5.2 Processing Threshold and Range Queries

Consider the MET query requesting the sequence pairs such that the covariance is greater than a user-defined threshold τ. We obtain the converted query by dividing τ by $\|\boldsymbol{\alpha}_q\|$. We call this the modified threshold $\tau' = \frac{\tau}{\|\boldsymbol{\alpha}_q\|}$. For computing the modified threshold τ' the value of $\boldsymbol{\alpha}_q$ corresponding to covariance in Table 3.2 is used. Note that the this conversion guarantees that the result set of the original and the converted query are the same. Next, we scan all the B-trees associated with all the pivot nodes, using a binary search algorithm and collect e_{qd}, such that $\tau' > \xi_{qd}$. Figure 3.7 shows an example of this process. The collected set of e_{qd} is the result set $\mathcal{A}_T$ of the MET query.

Secondly, consider the measure range query requesting all the sequence pairs, such that their covariance is in between thresholds τ_l and τ_u. Similar to the MET query, we compute the modified thresholds: $\tau_l' = \frac{\tau_l}{\|\boldsymbol{\alpha}_q\|}$ and $\tau_u' = \frac{\tau_u}{\|\boldsymbol{\alpha}_q\|}$. We, then, collect all the e_{qd} from all the B-trees using a binary search, such that $\tau_l' < \xi_{qd} < \tau_u'$. The collected set of e_{qd} is the result set $\mathcal{A}_R$ of the measure range query.

3.5.3 Index-based Pruning for D-Measures

Processing the MET and MER queries over the D-measures is a challenging problem. Recall that a D-measure is derived by normalizing a C-measure. The primary challenge is that normalization destroys the ordering of the scalar projections ξ_{qd}, which is established for processing queries for the L- and C-measures. Now, the idea here is to prune the sequence pairs stored in a sorted container using the values $\max(\varphi_q)$ and $\min(\varphi_q)$, stored in each pivot node. Our pruning technique quickly eliminates a large number of sequence pairs that do not satisfy the query condition(s).

Suppose we have a SCAPE index and a MET query that is requesting all sequence pairs such that the correlation coefficient, which is a D-measure, is greater than τ. We start the processing by considering each pivot node at a time. For a given pivot node, we compute the two modified thresholds: $\min(\tau') = \frac{\tau \cdot \min(\varphi_q)}{\|\boldsymbol{\alpha}_q\|}$ and $\max(\tau') = \frac{\tau \cdot \max(\varphi_q)}{\|\boldsymbol{\alpha}_q\|}$. Observe that the sequence nodes, associated to a pivot node, where $\xi_{pd} > \max(\tau')$, are definitely in the result set $\mathcal{A}_T$, and do not require further processing. This situation in depicted in Fig. 3.8(a) and holds because of the following:

$$\xi_{pd} > \max(\tau') \Leftrightarrow \frac{\|\boldsymbol{\alpha}_q\| \cdot \xi_{qd}}{\max(\varphi_q)} > \tau \Leftrightarrow \rho_{e_{qd}}(\mathbf{S}) > \tau. \tag{3.18}$$

Thus for all the sequence nodes where $\xi_{pd} > \max(\tau')$ the correlation coefficient can only be greater than τ.

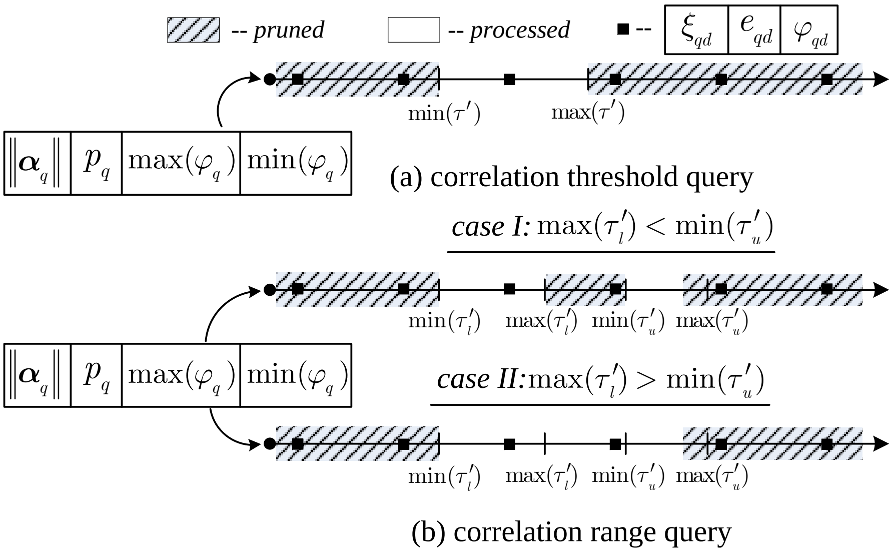

-- pruned
-- processed
ξ_{qd} e_{qd} φ_{qd}
$\min(\tau')$
$\max(\tau')$
$\|\alpha_q\|$ p_q $\max(\varphi_q)$ $\min(\varphi_q)$
(a) correlation threshold query
case I: $\max(\tau'_l) < \min(\tau'_u)$
$\min(\tau'_l)$ $\max(\tau'_l)$ $\min(\tau'_u)$ $\max(\tau'_u)$
case II: $\max(\tau'_l) > \min(\tau'_u)$
$\min(\tau'_l)$ $\max(\tau'_l)$ $\min(\tau'_u)$ $\max(\tau'_u)$
(b) correlation range query

Figure 3.8: Index-based pruning for processing MET and MER queries on D-measures.

Likewise, the correlation coefficient for all the sequence nodes where $\xi_{pd} < \min(\tau')$ can only be less than τ and can be excluded from the result set $\mathcal{A}_T$. The sequence nodes where $\min(\tau') < \xi_{qd} < \max(\tau')$ cannot be pruned. Thus, for these sequence nodes, we compute the correlation coefficient and check whether it is greater than τ and update the result set $\mathcal{A}_T$.

Similarly, consider a measure range query that is requesting all the sequence pairs such that their correlation coefficient is between τ_l and τ_u. As before, we compute four modified thresholds: $\min(\tau'_l) = \frac{\tau_l \cdot \min(\varphi_q)}{\|\boldsymbol{\alpha}_q\|}$, $\max(\tau'_l) = \frac{\tau_l \cdot \max(\varphi_q)}{\|\boldsymbol{\alpha}_q\|}$, $\min(\tau'_u) = \frac{\tau_u \cdot \min(\varphi_q)}{\|\boldsymbol{\alpha}_q\|}$, and $\max(\tau'_u) = \frac{\tau_u \cdot \max(\varphi_q)}{\|\boldsymbol{\alpha}_q\|}$. Again, following a similar reasoning as the MET query, the sequence nodes where $\xi_{pd} > \max(\tau'_u)$ and $\xi_{pd} < \min(\tau'_l)$ cannot be in the result set $\mathcal{A}_R$.

Furthermore, for the sequence nodes where $\max(\tau'_l) < \xi_{qd} < \min(\tau'_u)$ there could be two cases: (1) *case I*: $\max(\tau'_l) < \min(\tau'_u)$, and (2) *case II*: $\max(\tau'_l) > \min(\tau'_u)$. These cases are depicted in Fig. 3.8(b). For *case I*, the sequence nodes where $\max(\tau'_l) < \xi_{qd} < \min(\tau'_u)$ can be directly included in the result set $\mathcal{A}_R$ without further processing. In *case II*, pruning like case *case I* is not possible. In both the cases, for the unpruned sequence nodes we compute the correlation coefficient and check whether it is in between τ_l and τ_u and update the result set $\mathcal{A}_R$.

Note that the same index pruning techniques can be utilized for other D-measures. In Section 3.6, we compare the query processing methods using the SCAPE index with: (a) a method that uses affine relationships to compute the statistical measure and then process the MET or MER, and (b) a method that first computes the statistical measure from scratch and then processes the MET or MER query. Our experiments show that by using the SCAPE index structure we obtain a dramatic improvement in performance as compared to the other methods.

3.6 Experimental Evaluation

In this section we perform extensive experimental evaluation on real datasets to establish the efficacy of our approaches. In Section 3.6.1, we analyze the trade-off between accuracy and efficiency for computing statistical measures using affine relationships. We emphasize the performance improvements in query processing using synthetic – but realistic – workloads in Section 3.6.2. The scalability of the SYMEX algorithm is established in Section 3.6.3, and the performance gains by using the SCAPE index are demonstrated in Section 3.6.4. Since we have more than one method for computing and querying the statistical measures, as a shorthand we use the following notations:

- W_N: the naive method that computes a given statistical measure from scratch,
- W_A: the affine relationships method that uses affine relationships for computing a statistical measure (refer Section 3.4.1),
- W_F: an approach that uses the five largest DFT (Discrete Fourier Transform) coefficients for approximating the correlation coefficient, and has been introduced in [81, 97, 137].

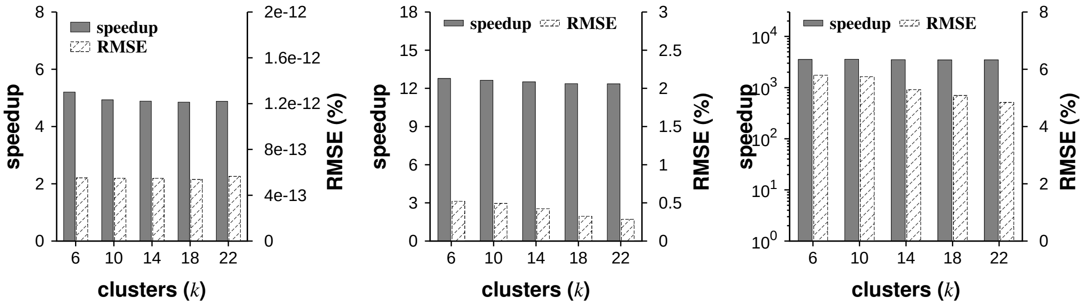
In this chapter we use two real datasets. The first dataset contains 670 daily time series obtained from 134 sensors monitoring environmental parameters on a university campus. We refer to this dataset as *sensor-data*. The second dataset consists of weekly, intra-day stock quotes from 996 stocks from the S&P 500 index and ETFs (exchange traded funds). We refer to this dataset as *stock-data*. The most important characteristics of the datasets are summarized in Table 3.3.

Table 3.3: Summary of the datasets.

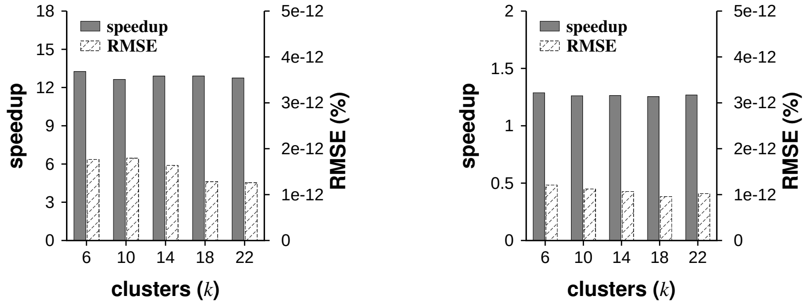
	sensor-data	*stock-data*
sampling interval	2 min.	1 min.
#time series (n)	670	996
#samples per time series (m)	720	1,950
max. affine relationships	224,115	495,510

3.6.1 Analyzing Trade-Off

For analyzing the tradeoff between efficiency and accuracy we consider a MEC query that computes a statistical measure (L, C, or D) over *all* the time series present in a dataset. Figure 3.9 and Figure 3.10 show the speedup and the percentage RMSE error (refer Section 3.4.1) obtained for all the statistical measures as a function of the number of affine clusters k. The speedup is computed as the ratio of time taken by the W_N method as compared to the W_A method. To give a sense of the absolute times, in Figure 3.11 we show the absolute time comparison for *stock-data*.



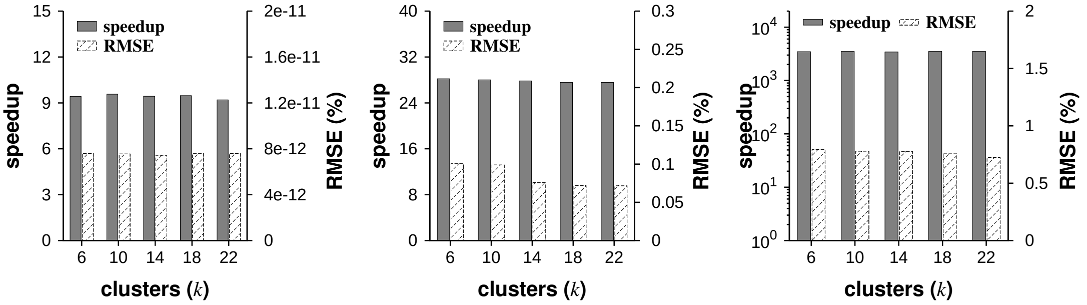
(a) mean (b) median (c) mode



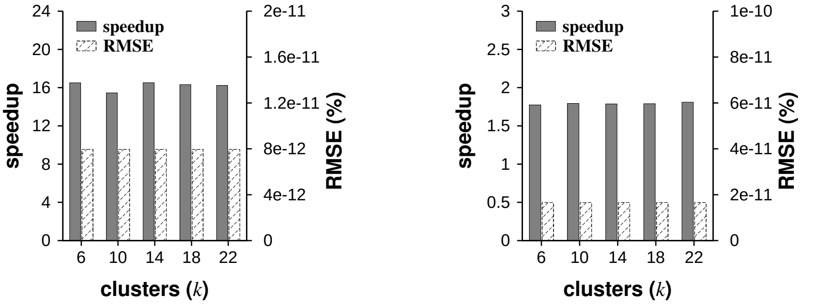
(d) covariance (e) dot product

Figure 3.9: Efficiency and accuracy tradeoff for *sensor-data*. Note the logarithmic scale for the speedup in (c).

In particular, for computing statistical measures, the focus of our work, the errors are negligible. The speedup obtained over all the statistical measures varies largely from a *factor 1.3 to 3500*. The maximum speedup of approximately *3500 times* is obtained for mode and the minimum speedups of *1.3x* and *4x* are obtained for dot product and mean respectively. The speedup obtained for mean and dot product is low due to the

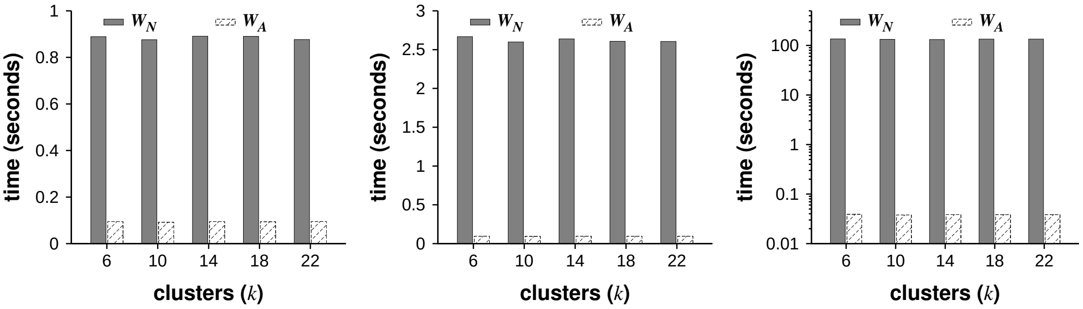


(a) mean (b) median (c) mode

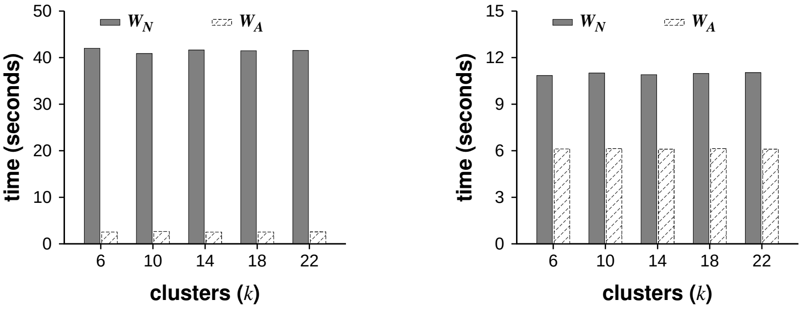


(d) covariance (e) dot product

Figure 3.10: Efficiency and accuracy tradeoff for *stock-data*. Note the logarithmic scale for the speedup in (c).



(a) mean (b) median (c) mode



(d) covariance (e) dot product

Figure 3.11: Absolute time comparison for *stock-data*. Note the logarithmic scale for the speedup in (c).

inherent simplicity of computing them using the W_N method. Thus, in summary, the W_A method exhibits significant improvements in efficiency and accuracy.

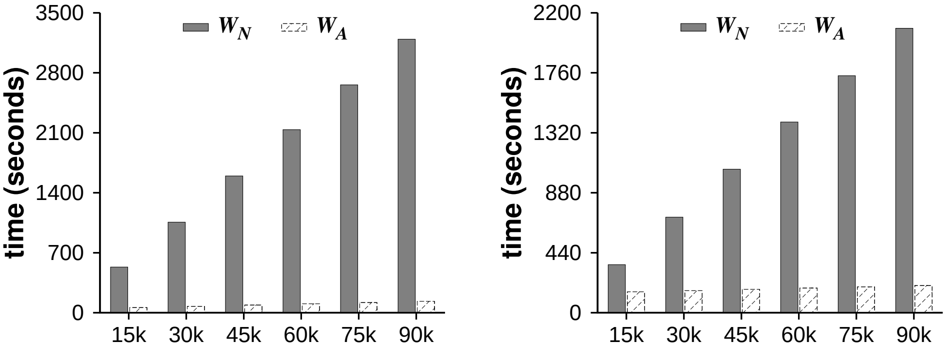
Since the *stock-data* is larger than the *sensor-data*, the efficiency improvement for *stock-data* is more prominent than *sensor-data*. This demonstrates that our approaches are capable of effectively handling large datasets. Moreover, for all the statistical measures a small number of clusters (6) are sufficient to obtain high accuracy; thus resulting in a nearly linear cost of processing the MEC query.

3.6.2 Impact of Online Environments

Our task here is to analyze how the Affinity framework handles MEC queries posed in online environments. Typically, in online environments, users frequently request for computation of a particular statistical measure for only few entities (stocks or sensors). To simulate this behavior, we generate realistic query workloads as follows: each query chooses uniformly at random a L-, C-, or D-measure and uses a powerlaw distribution for choosing 10 different series identifiers to form the set ψ. The intuition behind the powerlaw distribution is that since some entities (stocks or sensors) are popular as compared to others, thus we model their popularity with a powerlaw distribution.

Figure 3.12 compares query processing performance as the number of queries increase for the *sensor-data* and the *stock-data*. Here the parameters of the SYMEX+ algorithm are chosen as: $k = 6$, $\max(\gamma) = 10$, and $\min(\delta) = 10$. The gains obtained by using the W_A method are many-fold as compared to the W_N method. For example, the W_A method is *10 to 23 times faster* as compared to the W_N method when 90k queries are processed, and it is *2.5 to 9 times* faster when 15k queries are processed. Note that the time for the W_A method shown in Figure 3.12 also includes the initial time taken by the SYMEX+ algorithm for computing the affine relationships.

Thus, the proposed W_A method is far superior than the W_N method of query processing and is suitable for deployment in online environments. Here we cannot compare



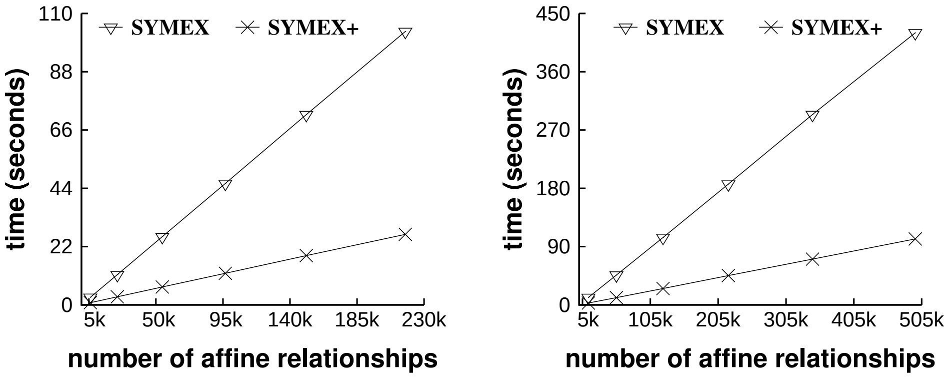
(a) *sensor-data* (b) *stock-data*

Figure 3.12: Comparing query processing efficiency.

with the W_F method, since the W_F method only computes the correlation coefficient and does not work for all the other statistical measures.

3.6.3 Scalability of the SYMEX Algorithm

Now we compare scalability of the SYMEX and SYMEX+ algorithms when the number of affine relationships generated by them increases. Figure 3.13 shows the scaling behavior of the SYMEX and the SYMEX+ algorithms as the number of affine relationships handled by these algorithms increase. For experiments in Figure 3.13, we set $k = 6$, $\max(\gamma) = 10$, and $\min(\delta) = 10$ as the parameters of the AFCLST algorithm. The SYMEX and the SYMEX+ algorithms scale linearly as the number of affine relationships increase. Particularly, the SYMEX+ algorithm is a factor *3.5 to 4 times* faster as compared to the simple SYMEX algorithm. Thus, the SYMEX+ algorithm exhibits attractive improvements as compared to the SYMEX algorithm.



(a) (b)

Figure 3.13: Scalability of the SYMEX algorithm. (a) *sensor-data* and (b) *stock-data.*

3.6.4 Impact of using the SCAPE Index

We discuss the performance improvements obtained by using the SCAPE index. Here, all the experiments are performed on the *sensor-data*. Recall, the SCAPE index uses the affine relationships returned by the SYMEX+ algorithm. For processing the MET and MER queries on the correlation coefficient the index pruning methods discussed in Section 3.5.3 are utilized.

We first analyze the scalability of constructing the SCAPE index as the number of indexed affine relationships increase. Figure 3.14 shows the scaling behavior of the SCAPE index when it indexes the affine relationships for a C-measure (covariance) and a L-measure (mean). Clearly, the SCAPE index exhibits linear scaling, which makes it a viable practical alternative for query processing.

Next, we compare the performance improvement obtained by using the SCAPE index for processing the MET and MER queries for the covariance and the correlation

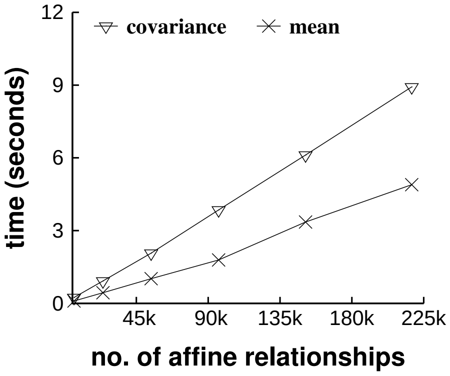
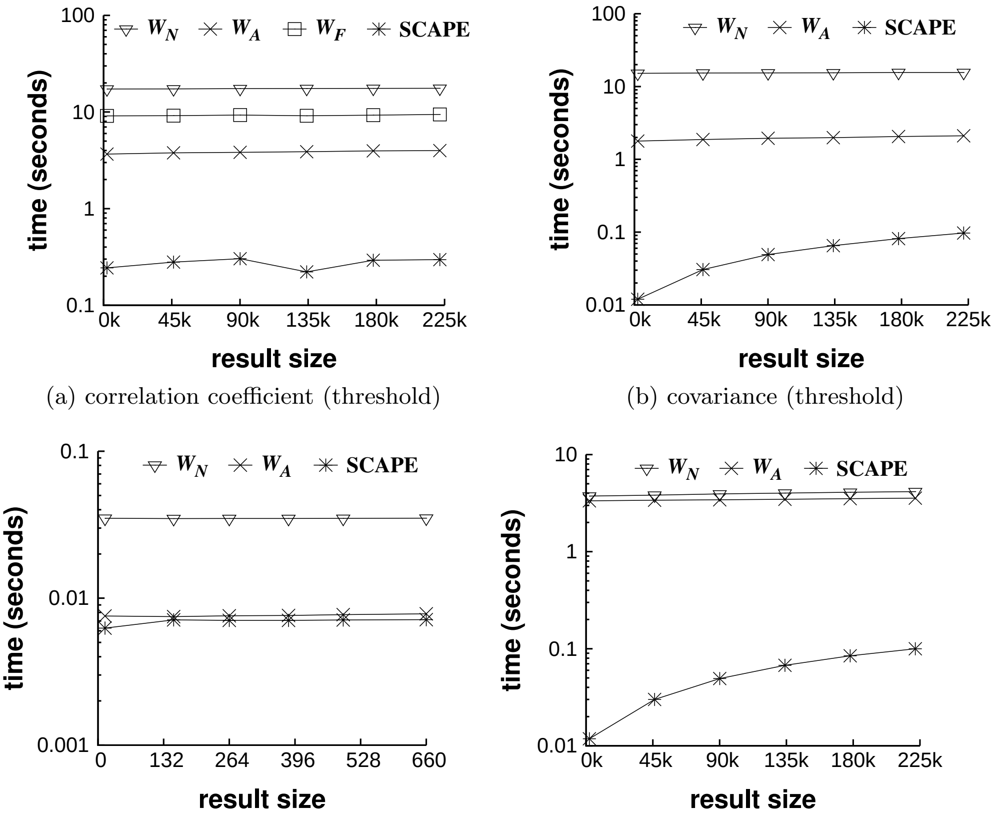


Figure 3.14: Scalability of the index construction on *sensor-data*.



(a) correlation coefficient (threshold)

(b) covariance (threshold)

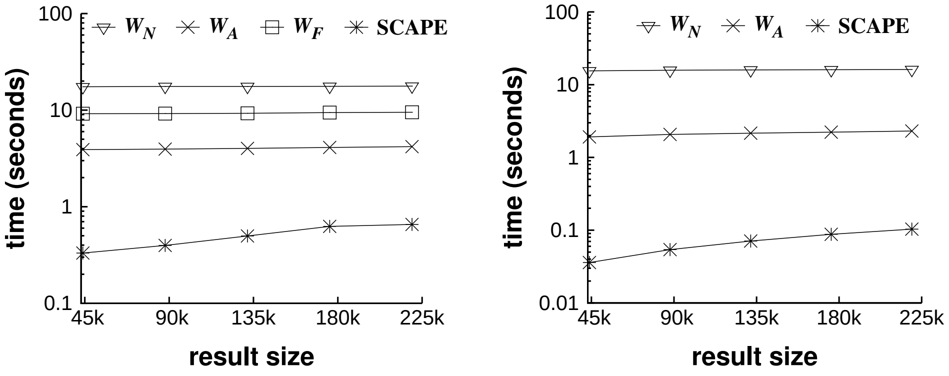
(c) median

(d) dot product

Figure 3.15: Comparing efficiency of the SCAPE index for the MET query.

coefficient. Here, all the affine relationships that are returned by the SYMEX+ algorithm are used for creating the SCAPE index. Figure 3.15 and Figure 3.16 compares the results for query processing obtained using the SCAPE index with the other methods. The other methods (W_N, W_A, and W_F) first compute the required statistical measure and then trivially evaluate the MET or MER query. Note that since W_F only computes the correlation coefficient, therefore it is only include in Fig. 3.15(a) and Fig. 3.16(a).

Figure 3.15 and Figure 3.16 depict the orders of magnitude improvement (shown



(a) correlation coefficient (b) covariance

Figure 3.16: Comparing efficiency of the SCAPE index for the MER query.

using logarithmic scale) in efficiency while processing the MET and MER queries using the SCAPE index. Table 3.4 shows a snapshot of the orders of magnitude performance improvement for all the statistical measures, and also in particular when comparing to the best known methods from the literature (W_F) for the computation of the correlation coefficient.

It is clearly evident from Table 3.4, that by using the SCAPE index we obtain orders of magnitude performance improvement. There is, however, one exception – median. Since median is a L-measure, the maximum possible number of affine relationships for it are low (linear in n). These affine relationships are insufficient for demonstrating the efficacy of the SCAPE index. In summary, the proposed indexing methods exhibit tremendous improvement in the efficiency of processing MET and MER queries.

Table 3.4: Query processing speedup computed when the query returns the maximum size of the result set $\mathcal{A}_T$ or $\mathcal{A}_R$.

Query type	Measure	Speedup		
		W_N	W_A	W_F
MET	correlation coefficient	59x	13.4x	32x
	covariance	160x	21x	×
	dot product	41x	35x	×
	median	5x	1.1x	×
MER	correlation coefficient	27x	6.4x	14x
	covariance	155x	22x	×

3.7 Related Work

Many prior works transform data from time domain to frequency domain using the DFT and then use the equivalence of norms (Parseval's theorem) property of the DFT for approximating the correlation coefficient using the largest DFT coefficients [81, 97, 137]. Computing the Pearson's correlation coefficient using DFT-based techniques provides inaccurate results when the time series contain white noise. Cole *et al.* [31] call such time series *uncooperative* and propose methods for discovering correlation amongst such signals. All these studies, however, typically only consider the correlation coefficient and do not propose an unified approach for computing and querying a wide variety of statistical measures, which includes the correlation coefficient.

In addition to computing the correlation coefficient, there has been a large body of related prior research using the DFT for: (a) exact or approximate sequence matching where the sequences could have undergone a similarity transformation [6, 7, 46], (b) retrieving similar shapes [59, 108], (c) predicting future values and answering similarity queries [83], and (d) reducing the dimensionality of the time-series data [69, 69]. Our work, on the contrary, considers affine transformations, which are a more generalized form of similarity transformations. Secondly, these techniques do not notice that affine transformations can be used for efficiently computing statistical measures.

TAPER [132] defines an all-strong-pairs correlation query that returns pairs of highly positively correlated items given a user-specified threshold. SPRIT [102], on the other hand, uses PCA (Principal Component Analysis) for summarizing a large collections of streams and discovering correlations. Our work differs from those mainly due to the fact that those techniques are tightly coupled to a particular type of query or statistical measure, most often the correlation coefficient. In that sense our work is unique.

Processing aggregate or related queries over time-series data is another area related to our work. A method of computing correlated aggregates is proposed in [51]. The Cypress framework [111] uses Fourier transform and random projection based multi-scale analysis for segmenting the data into various form of trickles, which are then used for query processing. Similarly, GAMPS [48] uses ratio signals for compressing time-series data and proposes approaches for query processing over such compressed data. More recently, there has been research conducted on indexing and querying correlated uncertain information using probabilistic databases [67, 110]. Lastly, Ke *et al.* [68] propose approaches for searching graphs correlated to a given query graph.

3.8 Conclusion

In this chapter, for the first time, we defined and proposed the notion of affine relationships for computing and querying several statistical measures using an unified approach. We proposed the affine clustering algorithm for clustering the time-series data, such that high-quality affine relationships could be found. We showed that using affine relationships results in dramatic performance improvement in computing statistical measures with minimal loss in accuracy. We demonstrated that the SCAPE index structure can easily index all the statistical measures and produce orders of magnitude improvement in efficiency for processing measure threshold and range queries, as compared to naive methods and methods proposed in the literature for this problem. In the next chapter, we will consider the problem of characterizing uncertainty in time-series data. As a solution to this problem, we will provide methods for estimating evolving probability distributions that effectively capture uncertainty in time-series data.

Chapter 4

Creating Probabilistic Databases from Imprecise Time-Series Data

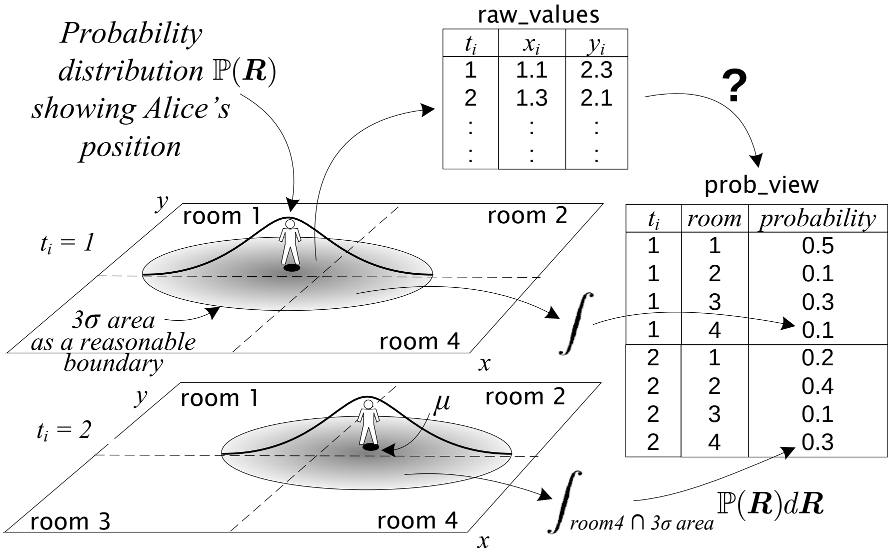
> *Les questions les plus importantes de la vie ne sont en effet, pour la plupart, que des problèmes de probabilité.*
> *(Life's most important questions are, for the most part, nothing but probability problems.)*
>
> Pierre-Simon Laplace

4.1 Introduction

In this chapter, we propose methods for characterizing uncertainty in imprecise time-series data. In recent years there have been a plethora of methods for managing and querying uncertain data [24, 32, 34, 57, 100, 110, 124]. These methods are typically based on the assumption that probabilistic data used for processing queries is available; however, this is not always true. Creating probabilistic data is a challenging and still unresolved problem. Prior work on this problem has only limited scope for domain-specific applications, such as handling duplicated tuples [10, 56] and deriving structured data from unstructured data [55]. Evidently, a wide range of applications still lack the benefits of existing query processing techniques that require probabilistic data. Time-series data is one important example where probabilistic data processing is currently not widely applicable due to the lack of probability values. Although, the benefits are evident given that time series, in particular generated from sensors (environmental sensors, RFID, GPS, etc.), are often imprecise and uncertain in nature.

Before diving into the details of our approach let us consider a motivating example shown in Figure 4.1. Here, Alice is tracked by indoor-positioning sensors and her locations are recorded in a database table called `raw_values` in the form of a three-tuple

(t_i, x_i, y_i). These raw values are generally imprecise and uncertain due to several noise factors involved in position measurement, such as low-cost sensors, discharged batteries, and network failures. On the other hand, consider a probabilistic query where an application is interested in knowing, given a particular time, the probability that Alice could be found in each of the four rooms. For answering this query we need the table `prob_view` (see Figure 4.1). This table gives us the probability of finding Alice in a particular room at a given time. To derive the `prob_view` table from the `raw_values` table, however, the system faces a fundamental problem—how to *meaningfully* associate a probability distribution $\mathbb{P}(\boldsymbol{R})$ with each raw value tuple (t_i, x_i, y_i), where $\boldsymbol{R}$ is the random variable associated with Alice's position.


Probability distribution $\mathbb{P}(\boldsymbol{R})$ showing Alice's position
raw_values
?
prob_view
$t_i = 1$
$t_i = 2$
y
x
room 1
room 2
room 3
room 4
3σ area as a reasonable boundary
μ
$\int$
$\int_{room4 \cap 3\sigma\ area} \mathbb{P}(\boldsymbol{R}) d\boldsymbol{R}$

t_i	x_i	y_i
1	1.1	2.3
2	1.3	2.1
⋮	⋮	⋮

t_i	room	probability
1	1	0.5
1	2	0.1
1	3	0.3
1	4	0.1
2	1	0.2
2	2	0.4
2	3	0.1
2	4	0.3

Figure 4.1: An example of creating a tuple-level probabilistic database from time-dependent probability distributions.

Once the system associates a probability distribution $\mathbb{P}(\boldsymbol{R})$ with each tuple, it can be used to derive probabilistic views, which forms a probabilistic database used for evaluating various types of probabilistic queries [24, 34]. Thus, this example clearly illustrates the importance of having a means for creating probabilistic databases. Nevertheless, there is a lack of effective tools that are capable of creating such probabilistic databases. In an effort to rectify this situation, we focus on the problem of creating a probabilistic database from given (imprecise) time series, thereupon, facilitating direct processing of a variety of probabilistic queries.

Unfortunately, creating probabilistic databases from imprecise time-series data poses several important challenges. In the following paragraphs we elaborate these challenges and discuss the solutions that this chapter proposes.

Inferring Evolving Probability Distributions.

One of the most important challenges in creating a probabilistic database from time series is to deal with evolving probability distributions, since time series often exhibit highly irregular dependencies on time [32, 126]. For example, temperature changes dramatically around sunrise and sunset, but changes only slightly during the night. This implies that the probability distributions that are used as the basis for deriving probabilistic databases also change over time, and thus must be computed dynamically.

In order to capture the evolving probability distributions of time series we introduce various *dynamic density metrics*, each of them dynamically infers time-dependent probability distributions from a given time series. The distributions derived by these dynamic density metrics are then used for creating probabilistic databases. After carefully analyzing several dynamical models for representing the dynamic density metrics (details are provided in Section 4.3 and Section 4.7), we identify and adopt a novel class of dynamical models from the time-series literature, which is known as the GARCH (Generalized AutoRegressive Conditional Heteroskedasticity) model [120]. We show that the GARCH model can play an important role in efficiently and accurately creating probabilistic databases, by effectively inferring dynamic probability distributions.

An important challenge in identifying appropriate dynamic density metrics is to find a measure that precisely assess the quality of the probability distributions produced by these metrics. This assessment is important since it quantifies the quality of probabilistic databases derived using these probability distributions. A straightforward method is to compare the ground truth (i.e., true probability distributions) with the inference obtained from our dynamic density metrics, thus producing a tangible measure of quality. This is, however, infeasible since we can neither observe the ground truth nor establish it unequivocally by any other means. To circumvent this crucial limitation, we propose an indirect method for measuring quality, termed *density distance*, which is based on a solid mathematical framework. The density distance is a generic measure of quality, which is independent of the models used for producing probabilistic databases.

Unfortunately, the GARCH model works inappropriately on time series that contain erroneous values, i.e., significant outliers, which are often produced by sensors. This is because the GARCH model is generally used over precise, certain, and clean data (e.g., stock market data). In contrast, the time series that this study considers are typically imprecise and erroneous. Thus, we propose an improved version of the GARCH model, termed C-GARCH, that performs appropriately in the presence of such erroneous values.

Efficiently Creating Probabilistic Databases.
Given probability distributions inferred by a dynamic density metric, the next step of our solution is to generate views that contain probability values (e.g., `prob_view` in Figure 4.1). We introduce the Ω-*View builder* that efficiently creates probabilistic views by processing a *probability value generation query*. The output of this query can be directly consumed by a wide variety of existing probabilistic queries, thus enabling higher level probabilistic reasoning.

Since the probabilistic value generation query accepts arbitrary time intervals (past or current) as inputs, this could incur heavy computational overhead on the system when the time interval spans over a large number of raw values. To address this, we present an effective caching mechanism called *σ-cache.* The σ–cache caches and reuses probability values computed at previous times for current time processing. We experimentally demonstrate that the σ–cache boosts the efficiency of query processing by an *order of magnitude.* Additionally, we provide theoretical guarantees that are used for setting the cache parameters. These guarantees enable the choice of the cache parameters under user-defined constraints of storage space and error tolerance. Moreover, such guarantees make the σ–cache an attractive solution for large-scale data processing.

4.1.1 Chapter Organization

We begin by giving details of our framework for generating probabilistic databases in Section 4.2. Section 4.3 introduces the naive dynamic density metrics while in Section 4.4 we propose the GARCH metric. An enhancement of the GARCH metric, C-GARCH, is discussed in Section 4.5. In Section 4.6, we suggest effective methods for generating probabilistic databases, this is followed by a discussion on σ–cache. Lastly, Section 4.7 presents comprehensive experimental evaluations followed by the review of related studies in Section 4.8.

4.2 Foundation

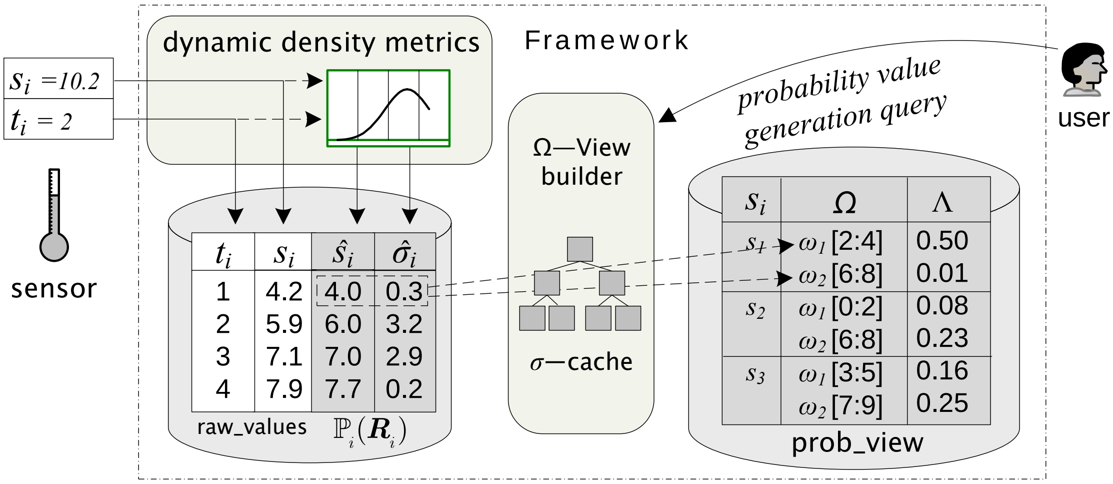
This section describes our framework, defines queries this study considers, and proposes a measure for quantifying the effectiveness of the dynamic density metrics. Table 4.1 offers the notations used in this chapter.

4.2.1 Framework Overview

Figure 4.2 illustrates our framework for creating probabilistic databases, consisting of two key components that are dynamic density metrics and the Ω–View builder. A *dynamic density metric* is a system of measure that dynamically infers time-dependent probability distributions of imprecise raw values. It takes as input a sliding window that contains recent previous values in the time series. In the following sections, we introduce various dynamic density metrics.

Unlike the setup that we considered in Chapter 2, a large part of this chapter only considers the problem of creating probabilistic databases from time-series data obtained from a single data source. Therefore, in the notation used in this chapter we drop the subscript j that was used to identify the data source in Chapter 2. Moreover, we consider the problem of creating probabilistic databases from multivariate time-series data in [65].

We start by denoting the time series as by a vector $\boldsymbol{s} = [s_1, s_2, \cdots, s_l]$. Each element of the time series is represented as s_i where $1 \leq i \leq l$. s_i indicates a (imprecise) raw value at time t_i. Similar to the time-series database model in Section 1.1, we consider the sampling interval to be uniform, that is, $t_{i+1} - t_i$ is same for all the values of $i \geq 1$. Therefore, for simplifying the notation we denote the time axis only with the index i.



t_i	s_i	$\hat{s}_i$	$\hat{\sigma}_i$
1	4.2	4.0	0.3
2	5.9	6.0	3.2
3	7.1	7.0	2.9
4	7.9	7.7	0.2

s_i	Ω	Λ
s_1	ω_1 [2:4]	0.50
	ω_2 [6:8]	0.01
s_2	ω_1 [0:2]	0.08
	ω_2 [6:8]	0.23
s_3	ω_1 [3:5]	0.16
	ω_2 [7:9]	0.25

Figure 4.2: Architecture of the framework.

Unlike Chapter 2, the time series $\boldsymbol{s}$ defined here does not have a fixed size n. As new time-series data streams into the system, the size of $\boldsymbol{s}$ increases. Let $\boldsymbol{s}^H_{l-1} = [s_{l-H}, s_{l-H+1}, \cdots, s_{l-1}]$ be a (sliding) window that is a subsequence of $\boldsymbol{s}$, where its ending value is at the previous time of l. The dynamic density metrics correspond to the following query:

Definition 4.1: **Inference of dynamic probability distribution.** Given a (sliding) window $\boldsymbol{s}^H_{t-1}$, the inference of a probability distribution at time l estimates a probability density function $\mathbb{P}_l(R_l)$, where R_l is a random variable associated with s_l.

The system stores the inferred probability density functions $\mathbb{P}_l(R_l)$ associated with the corresponding raw values. Next, our Ω–View builder uses these inferred probability density functions to create a probabilistic database, as shown in the `prob_view` table of Figure 4.2.

Suppose that the data values of a probabilistic database are decomposed into a set of ranges $\Omega = \{\omega_1, \omega_2, \cdots, \omega_U\}$, where $\omega_u = [\min(\omega_u), \max(\omega_u)]$ is bounded by a lower bound $\min(\omega_u)$ and an upper bound $\max(\omega_u)$ for $1 \le u \le U$. Then, the Ω–View builder corresponds to the following query in order to compute probability values for the given ranges:

Definition 4.2: **Probability value generation query.** Given a probability density function $\mathbb{P}_l(R_l)$ and a set of ranges $\Omega = \{\omega_1, \omega_2, \cdots, \omega_U\}$ for the probability values in a probabilistic database, a probability value generation query returns a set of probabilities $\Lambda_l = \{o_1, o_2, \cdots, o_U\}$ at time l, where o_u is the probability of occurrence of $\omega_u \in \Omega$ and is equal to $\int_{\min(\omega_u)}^{\max(\omega_u)} \mathbb{P}_l(R_l) dR_l$, and $1 \le u \le U$.

Recall the example shown in Figure 4.1. Let us assume that ω_1 corresponds to the event of Alice being present in *Room 1*. At time $l = 1$, Alice is likely to be in *Room 1* (i.e., ω_1 occurs) with probability $o_1 = 0.5$.

Note that the creation of probabilistic databases can be performed in either online or offline fashion. In the online mode, the dynamic density metrics infer $\mathbb{P}_l(R_l)$ as soon as a new value s_l is streamed to the system. In the offline mode, users may give SQL-like queries to the system (examples are provided in Section 4.6).

Table 4.1: Summary of Notations.

Symbol	**Description**
$\boldsymbol{s}$	A time series.
$\boldsymbol{s}_{l-1}^{H}$	Sliding window having H values in the range $[l-H, l-1]$.
s_l	Raw (imprecise) value at time l.
R_l	Random variable associated with s_l.
$\hat{s}_l, \mathrm{E}(R_l)$	Expected true value at time l.
$\mathbb{P}_l(R_l)$	Probability density function of R_l at time l.
$\mathrm{P}_l(R_l)$	Cumulative probability distribution function of R_l at time l.
o	Probability of occurrence of event ω.
$\mathrm{E}(X)$	Expected value of random variable X.
$\mathcal{N}(\mu, \sigma^2)$	Gaussian probability density function with mean μ and variance σ^2.
Ω	A set of ranges for creating probability values in a probabilistic database.
$\lceil x \rceil$	A smallest integer value that is not smaller than x.

4.2.2 Evaluation of Dynamic Density Metrics

Quantifying the quality of a dynamic density metric is crucial, since it reflects the quality of a probabilistic database created. Here, we introduce an effective measure, termed *density distance*, that quantifies the quality of a probability density inferred by a dynamic density metric.

Let $\mathbb{P}_l(R_l)$ be an inferred probability density at time l. A straightforward manner in which we can evaluate the quality of this inference is to compare $\mathbb{P}_l(R_l)$ with its corresponding true density $\hat{\mathbb{P}}_l(R_l)$. $\hat{\mathbb{P}}_l(R_l)$, however, cannot be given nor observed, rendering this straightforward evaluation infeasible. To overcome this, we propose to use an indirect method for evaluating the quality of a dynamic density metric known as the *probability integral transform* [41]. A probability integral transform of a random variable X, with probability density function $\mathbb{P}(X)$, transforms X to a uniformly distributed random variable Y by evaluating $Y = \int_{-\infty}^{x} \mathbb{P}(X = u)du$ where $x \in X$. Thus, the probability integral transform of s_i with respect to $\mathbb{P}_i(R_i)$ becomes, $y_i = \int_{-\infty}^{s_i} \mathbb{P}_i(R_i = u)du.$, where $1 \leq i \leq l$. Let $\mathbb{P}_1(R_1), \ldots, \mathbb{P}_l(R_l)$ be the sequence of probability distributions inferred using a dynamic density metric. Also, let $y_1, \ldots, y_l$ be the probability integral transforms of raw values $s_1, \ldots, s_l$ with respect to $\mathbb{P}_1(R_1), \ldots, \mathbb{P}_l(R_l)$. Then, $y_1, \ldots, y_l$ are uniformly distributed between $(0, 1)$ if and only if the inferred probability density $\mathbb{P}_i(R_i)$ is equal to the true density $\hat{\mathbb{P}}_i(R_i)$ for $1 \leq i \leq l$ [41].

To find out whether $y_1, \ldots, y_l$ follow a uniform distribution we estimate the cumulative distribution function of $y_1, \ldots, y_l$ using a histogram approximation method. Let us denote this cumulative distribution function as $\mathsf{Q}_Y(y)$. We define the quality measure of a dynamic density metric as the Euclidean distance between $\mathsf{Q}_Y(y)$ and the ideal uniform cumulative distribution function between $(0, 1)$ denoted as $\mathsf{U}_Y(y)$. Formally, the quality measure is defined as:

$$\|\mathsf{U}_Y(y) - \mathsf{Q}_Y(y)\|_2 = \sqrt{\sum_{x=0}^{1} (\mathsf{U}_Y(x) - \mathsf{Q}_Y(x))^2}. \tag{4.1}$$

We refer to $\|\mathsf{U}_Y(y) - \mathsf{Q}_Y(y)\|_2$ as *density distance*. The density distance quantifies the difference between the observed distribution of $y_1, \ldots, y_l$ and their expected distribution. Thus, it gives a measure of quality for the inferred densities $\mathbb{P}_1(R_1), \ldots, \mathbb{P}_l(R_l)$. The density distance will be used in Section 4.7 to compare the effectiveness of each dynamic density metrics this chapter introduces.

4.3 Naive Dynamic Density Metrics

This section presents two relatively simple dynamic density metrics that capture evolving probability densities in time series.

Uniform Thresholding Metric.
Cheng *et al.* [24, 27] have proposed a generic query evaluation framework over imprecise data. The key idea in these studies is to model a raw value as a user-provided uncertainty range in which the corresponding unobservable true value resides. Queries are then evaluated over such uncertainty ranges, instead of the raw values.

Our *uniform thresholding metric* extends this idea for estimating probability distributions by inferring a true value. We define such a true value as:

Definition 4.3: **Expected true value.** Given a probability density function $\mathbb{P}_l(R_l)$, the expected true value $\hat{s}_l$ is the expected value of R_l, denoted as $\mathrm{E}(R_l)$.

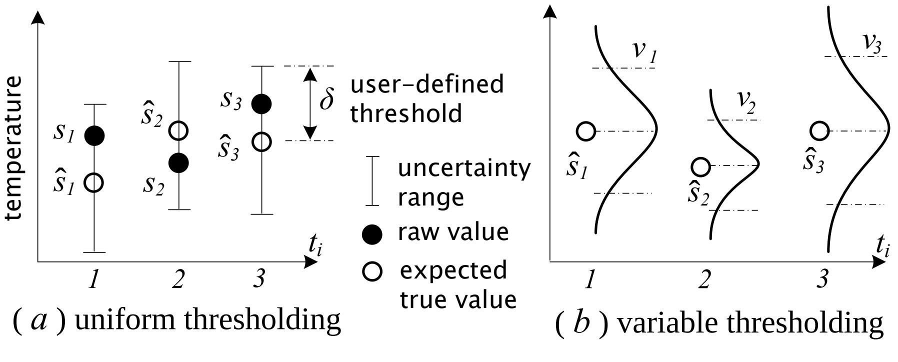


Figure 4.3: Examples of naive dynamic density metrics.

Next, the uniform thresholding metric takes a user-defined threshold value δ to bound uniform distributions, centered on the inferred true value. Figure 4.3 (a) illustrates an example of this process where a user-defined threshold value δ is used for specifying the uncertainty ranges. The difference between a true value $\hat{s}_l$ and its corresponding raw value s_l is then assumed to be not greater than δ.

To infer expected true values, we adopt the *AutoRegressive Moving Average* (ARMA) model [120] that is commonly used for predicting expected values in time series [127]. Specifically, given the time series $\boldsymbol{s} = [s_1, s_2, \cdots, s_l]$ and a sliding window $\boldsymbol{s}_{l-1}^{H}$, the ARMA model models $s_i = \hat{s}_i + e_i$, where $l - H \leq i \leq l - 1$ and e_i obeys a zero mean normal distribution with variance σ_e^2. Now, given an ARMA(α,β) model, we infer the expected true value $\hat{s}_l$ as:

$$\hat{s}_l = \Phi_0 + \sum_{j=1}^{\alpha} \Phi_j s_{l-j} + \sum_{j=1}^{\beta} \Theta_j e_{l-j}, \tag{4.2}$$

where (α, β) are non-negative integers denoting the model order, $\Phi_1, \ldots, \Phi_p$ are autoregressive coefficients, $\Theta_1, \ldots, \Theta_q$ are moving average coefficients, Φ_0 is a constant, and $l > max(\alpha, \beta)$. More details regarding the estimation and choice of the model parameters (α, β) are described in Chapter 3 in [120].

Variable Thresholding Metric.

We propose another dynamic density metric, termed *variable thresholding metric*, that differs in two ways from the uniform thresholding metric. First, the variable thresholding metric works on Gaussian distributions, while the uniform thresholding metric is applicable only to uniform distributions. Second, unlike the uniform thresholding metric, the variable thresholding metric does not require the user-defined threshold for specifying uncertainty ranges. Instead, it computes a sample variance ν_l^2 for a window $\boldsymbol{s}_{l-1}^{H}$, so that ν_l^2 is used to model a Gaussian distribution. Given $\boldsymbol{s}_{l-1}^{H}$, the variable thresholding metric infers a normal distribution at time l as:

$$\mathbb{P}_l(R_l = s_l) = \frac{1}{\sqrt{2\pi\nu_l^2}} e^{-(s_l - \hat{s}_l)^2/2\nu_l^2}, \tag{4.3}$$

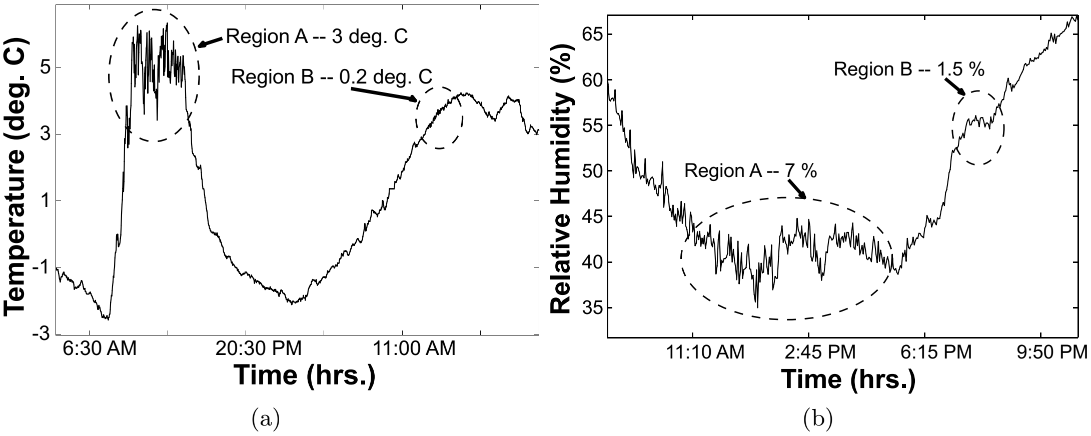
where $\hat{s}_l$ is an expected true value inferred by the ARMA model.

Figure 4.3 (b) demonstrates an example of estimating normal distributions based on the variable thresholding metric. First, the ARMA model infers the expected true values $\hat{s}_l$ that are used as the mean values for the normal distributions. It then computes the variances that are used to derive the standard deviations ν_l.

4.4 GARCH Metric

As stated in the previous section, it is common to capture the uncertainty of an imprecise time series with a fixed-size uncertainty range as shown in Figure 4.3 (a) [24, 27]. This approach, however, may not be effective in practice, since in a wide variety of real-world settings, the size of the uncertainty range typically varies over time. For example, Figure 4.4 shows two time series obtained from a real sensor network deployment monitoring ambient temperature and relative humidity. The regions marked as *Region A* in Figure 4.4 (*a*) and Figure 4.4 (*b*) exhibit higher volatility[1] than those marked as *Region B*. This observation strongly suggests that the underlying model should support time-varying variance and mean value when it infers a probability density function. We experimentally verify this claim in Section 4.7.4.

Motivated by this, we introduce a new dynamic density metric, the *GARCH metric*. The GARCH metric models $\mathbb{P}_l(R_l)$ as a Gaussian probability density function $\mathcal{N}(\hat{s}_l, \hat{\sigma}_l^2)$. This metric assumes that the underlying time series exhibits not only time-varying average behavior ($\hat{s}_l$) but also time-varying variance ($\hat{\sigma}_l^2$). For inferring $\hat{\sigma}_l^2$ we propose using the GARCH model. And, for inferring $\hat{s}_l$ we can either use the ARMA model from Section 4.3 or Kalman Filters.



(a) (b)

Figure 4.4: Regions of changing volatility in (a) ambient temperature and (b) relative humidity.

4.4.1 The GARCH Model

The GARCH (Generalized AutoRegressive Conditional Heteroskedasticity) model [120] efficiently captures time-varying volatility in a time series. Specifically, given a window $\boldsymbol{s}_{l-1}^H$, the ARMA model models $s_i = \hat{s}_i + e_i$ where $l - H \leq i \leq l - 1$.

We then define the conditional variance σ_i^2 as:

$$\sigma_i^2 = \mathrm{E}((s_i - \hat{s}_i)^2 | \mathcal{F}_{i-1}), \quad \sigma_i^2 = \mathrm{E}(e_i^2 | \mathcal{F}_{i-1}), \tag{4.4}$$

where $\mathrm{E}(e_i^2|\mathcal{F}_{i-1})$ is the variance of e_i given all the information $\mathcal{F}_{i-1}$ available until time $i-1$. The GARCH(ζ,η) model models volatility in Eq. (4.4) as a linear function of e_i^2 as:

$$e_i = \sigma_i \epsilon_i, \quad \sigma_i^2 = \Gamma_0 + \sum_{j=1}^{\zeta} \Gamma_j e_{i-j}^2 + \sum_{j=1}^{\eta} \Psi_j \sigma_{i-j}^2, \tag{4.5}$$

---

[1] We use *variance* and *volatility* interchangeably.

where ϵ_i is a sequence of independent and identically distributed ($i.i.d$) random variables, (ζ, η) are parameters describing the model order, $\Gamma_0 > 0$, $\Gamma_j \geq 0$, $\Psi_j \geq 0$, $\sum_{j=1}^{max(\zeta,\eta)}(\Gamma_j + \Psi_j) < 1$, and i takes values between $l - H + max(\zeta, \eta)$ and $l - 1$.

The underlying idea of the GARCH(ζ,η) model is to reflect the fact that large shocks (e_i) tend to be followed by other large shocks. Unlike the ν_l^2 in the variable thresholding metric, σ_i^2 is a variance that is estimated after subtracting the local trend $\hat{s}_i$. In many practical applications the GARCH model is typically used as the GARCH(1,1) model, since for a higher order GARCH model specifying the model order is a difficult task [120]. Thus, we restrict ourselves to these model order settings. More details regarding the estimation of model parameters and the choice for the sliding window size H are described in [120].

For inferring time-varying volatility, we use the GARCH(m,s) model and e_i as follows:

$$\hat{\sigma}_l^2 = \Gamma_0 + \sum_{j=1}^{\zeta} \Gamma_j e_{l-j}^2 + \sum_{j=1}^{\eta} \Psi_j \sigma_{l-j}^2. \tag{4.6}$$

Recall that we use the ARMA model for inferring the value of $\hat{s}_l$ given $\boldsymbol{s}_{l-1}^H$. We also consider the Kalman Filter [120] for inferring $\hat{s}_l$. We show the difference in performance between the Kalman Filter and the ARMA model in Section 4.7.1. Basically, the Kalman Filter models $\hat{s}_l$ using the following two equations,

$$\text{state equation: } \hat{s}_i = c_1 \cdot \hat{s}_{i-1} + z_{i-1}^1 \quad z_i^1 \sim \mathcal{N}(0, \sigma_{z^1}^2), \tag{4.7}$$

$$\text{observation equation: } s_i = c_2 \cdot \hat{s}_i + z_i^2 \quad z_i^2 \sim \mathcal{N}(0, \sigma_{z^2}^2), \tag{4.8}$$

where $\hat{s}_1$ is given a priori and c_1 and c_2 are constants. Since the GARCH model in Eq. (4.5) takes errors e_i as input, they are computed as $e_i = s_i - \hat{s}_i$ and are used by the GARCH model.

Considering both approaches for inferring $\hat{s}_l$ (ARMA model and Kalman Filter) we propose two dynamic density metrics, namely, *ARMA-GARCH* and *Kalman-GARCH*. Both of them use the GARCH model for inferring $\hat{\sigma}_l$. But for inferring $\hat{s}_l$ they use ARMA model and Kalman Filter respectively.

Algorithm 4.1 gives a concise description of the ARMA-GARCH metric. This algorithm uses the ARMA model for inferring $\hat{s}_l$ and the GARCH model for inferring $\hat{\sigma}_l^2$ (Step 3). The algorithm for Kalman-GARCH metric is the same as Algorithm 4.1, except that it uses the Kalman filter in Step 3 for inferring $\hat{s}_l$ instead of using the ARMA model. Here, $\kappa \geq 0$ is a scaling factor that decides the upper bound max(s_l) and the lower bound min(s_l). For example, when $\kappa = 3$, the probability that s_l lies between max(s_l) and min(s_l) is very high (approximately 0.9973).

The time complexities of the estimation step for the ARMA model and the GARCH model (Step 1 and 2) are $\mathcal{O}(H \cdot max(\alpha, \beta))$ and $\mathcal{O}(H \cdot max(\zeta, \eta))$ respectively [93]. Nevertheless, as the model order parameters are small as compared to H these estimation steps become significantly efficient.

Algorithm 4.1 Inferring $\hat{s}_l$ and $\hat{\sigma}_l^2$ using ARMA-GARCH.

```
Input: ARMA model parameters $(\alpha, \beta)$, sliding window $\boldsymbol{s}_{l-1}^{H}$, and scaling factor $\kappa$.
Output: Inferred $\hat{s}_l$, inferred volatility $\hat{\sigma}_l^2$, and $\kappa$-scaled bounds $\max(s_l), \min(s_l)$.
Estimate an $ARMA(\alpha, \beta)$ model on $\boldsymbol{s}_{l-1}^{H}$ and obtain $e_i$ where $l - H + max(\alpha, \beta) \leq i \leq l - 1$
Estimate a $GARCH(1,1)$ model using $e_i$'s
Infer $\hat{s}_l$ using $ARMA(\alpha, \beta)$ and $\hat{\sigma}_l^2$ using $GARCH(1,1)$
$\max(s_l) \leftarrow \hat{s}_l + \kappa\hat{\sigma}_l$ and $\min(s_l) \leftarrow \hat{s}_l - \kappa\hat{\sigma}_l$
return $\hat{s}_l$, $\hat{\sigma}_l^2$, $u_b$, and $l_b$
```

4.5 Enhanced GARCH Metric

In practice, time series often contain values that are erroneous in nature. For example, sensor networks, like weather monitoring stations, frequently produce erroneous values due to various reasons; such as loss of communication, sensor failures, etc. Unfortunately, the GARCH model is incapable of functioning appropriately when input streams contain such erroneous values. This is because the GARCH model has been generally used over precise, certain, and clean data (e.g., stock market data). To tackle this problem, we propose an enhancement of the GARCH metric, which renders the GARCH metric robust against erroneous time-series inputs.

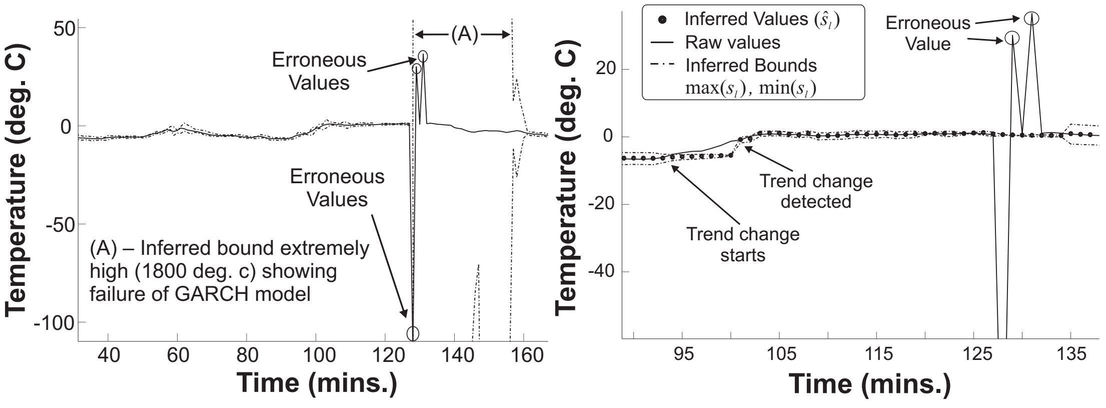
Before proceeding further, we note the difference between *erroneous values* and *imprecise values*. Imprecise values have an inherent element of uncertainty but still follow a particular trend, while erroneous values are significant outliers which exhibit large unnatural deviations from the trend.

To give an idea of the change in behavior exhibited by the GARCH model we run the ARMA-GARCH algorithm on all sliding windows $\boldsymbol{s}_{t-1}^{H}$ of a time series $\boldsymbol{s} = [s_1, s_2, \ldots, s_n]$ where $H + 1 \leq i \leq n$ and $\kappa = 3$. The result of executing this algorithm is shown in Figure 4.5 (a) along with the upper and lower bounds. Notice that at time 127, when the first erroneous value occurs in the training window, the GARCH model infers an extremely high volatility for the following time steps. This mainly happens since the GARCH equation Eq. (4.5) contains square terms, which significantly amplifies the effect of the presence of erroneous values. To avoid this we introduce novel heuristics which can be applied to input data in an online fashion and thus obtain a correct volatility estimate even in the presence of erroneous values. We term our approach C-GARCH (an acronym for Clean-GARCH).

4.5.1 C-GARCH Model

Let $\boldsymbol{s} = [s_1, s_2, \ldots, s_n]$ be a time series containing some erroneous values. We then start executing the ARMA-GARCH procedure (see Algorithm 4.1) at time $l > H$. For this we set $\kappa = 3$, thus making the probability of finding s_l outside the interval defined by $\max(s_l)$ and $\min(s_l)$ low. When we find that s_l resides outside $\max(s_l)$ and $\min(s_l)$, we mark it as erroneous value and replace it with the corresponding inferred value $\hat{s}_l$. Simultaneously, we also keep the track of the number of consecutive values we have marked as erroneous values most recently. If this number exceeds a predefined constant ς then we assume that the observed raw values are exhibiting a changing trend. For example, during sunrise the ambient temperature exhibits a rapid change of trend. This idea inherently assumes that the probability of finding ς consecutive erroneous values is low. And, if we find ς consecutive erroneous values we should re-adjust the model to the new trend.

Although it rarely happens in practice that there are many consecutive erroneous values may be present in raw data. To rule out the possibility of using these values for inference, we introduce a novel heuristic that is applied to the values in the window $[s_{l-\varsigma}, \ldots, s_l]$ before they are used for the inference. This step ensures that we have not included any erroneous values present in the raw data into our system. Thus we avoid the problems that occur by using a simple ARMA-GARCH metric.



(a) (b)

Figure 4.5: (a) Behavior of the GARCH model when window $\boldsymbol{s}_{l-1}^{H}$ contains erroneous values. (b) Result of using the C-GARCH model.

4.5.2 Successive Variance Reduction Filter

The heuristic that we use for filtering out significant anomalies is shown in Algorithm 4.2. This algorithm takes values $\boldsymbol{v} = [v_1, v_2, \ldots, v_K]$ containing erroneous values and a thresholding parameter $\max(\nu^2(\boldsymbol{v}))$ as input. It first measures dispersion of $\boldsymbol{v}$ by computing its sample variance denoted as $\nu^2(\boldsymbol{v})$ (Step 3) . Then we delete a point, say v_k, and compute the sample variance of all the other points $[v_1, \ldots, v_{k-1}, v_{k+1}, \ldots, v_K]$ denoted as $\nu^2(\boldsymbol{v} \backslash v_k)$ (Step 9). We perform this procedure for all points and then finally find a value $v_{\bar{k}}$, such that this value, if deleted, gives us the maximum variance reduction. We delete this point and reconstruct a new value at $\bar{k}$ using interpolation. We stop this procedure when the total sample variance becomes less than the variance threshold $\max(\nu^2(\boldsymbol{v}))$. In Steps 8 and 9, we use the intermediate values $\hat{v}'_K$ and $\hat{v}_K$ to compute $\nu^2(\boldsymbol{v} \backslash v_K)$, thus reducing the computational complexity of the algorithm to quadratic.

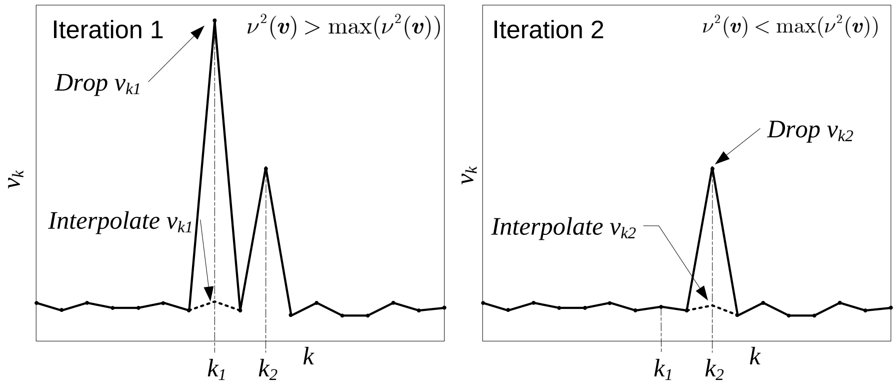


Figure 4.6: Showing sample run of the Successive Variance Reduction Filter (Algorithm 4.2).

A graphical example of our approach is shown in Figure 4.6. From this figure we can see that values at k_1 and k_2 are erroneous. In the first iteration our algorithm deletes value v_{k_1} and reconstructs it. Next, we delete v_{k_2} and obtain a new value using interpolation. At this point we stop since $\nu^2(\boldsymbol{v})$ becomes less than $\max(\nu^2(\boldsymbol{v}))$. Moreover, it is very important to know a fair value for $\max(\nu^2(\boldsymbol{v}))$, since if a higher value is chosen we might include some erroneous values and if a lower value is chosen we might delete some non-erroneous values. Also, the value of $\max(\nu^2(\boldsymbol{v}))$ depends on the underlying parameter monitored. For example, ambient temperature in Figure 4.4 shows rapid changes in trend as compared to relative humidity. Thus, using a sufficiently large sample of clean data, we compute $\max(\nu^2(\boldsymbol{v}))$ as the maximum sample variance (dispersion) we observe in all sliding windows of size ς. This gives a fair estimate of the threshold between trend changes and erroneous values.

Figure 4.5 (b) shows the result of using C-GARCH model on the same data as shown in Figure 4.5 (a) with $\varsigma = 7$. We can observe that at $l = 93$ a trend change starts to occur and is smoothly corrected by the C-GARCH model at $l = 101$. Most importantly, the successive variance reduction filter effectively handles the erroneous values occurring at times $l = 127$ and $l = 132$. Thus the C-GARCH model performs as expected and overcomes the shortcomings of the plain ARMA-GARCH metric. In Section 4.7 we will demonstrate the efficacy of the C-GARCH model on real data obtained from sensor networks.

Guidelines for Parameter Setting: The C-GARCH model requires three parameters κ, $\max(\nu^2(\boldsymbol{v}))$, and ς. In most cases we assign $\kappa = 3$. As seen before, $\max(\nu^2(\boldsymbol{v}))$ is learned from a sample of clean data. On the contrary, setting ς requires domain knowledge about sensors used for data gathering. If there are unreliable sensors which frequently emit erroneous values then setting a higher value for ς is advisable and vice versa. Our experiments suggest that the C-GARCH model performs satisfactorily when the value for ς is set to twice the length of the longest sequence of erroneous values. In practice, ς is generally small, making the execution of Algorithm 4.2 efficient.

Algorithm 4.2 The Successive Variance Reduction Filter.

```
Input: A time series $\boldsymbol{v}$ containing erroneous values and variance threshold $\max(\nu^2(\boldsymbol{v}))$.
Output: Cleaned values $\boldsymbol{v}$.
while true do
    $\hat{v}'_K \leftarrow \sum_{k=1}^{K} v_k^2$ and $\hat{v}_K \leftarrow \frac{1}{K}\sum_{k=1}^{K} v_k$
    $\nu^2(\boldsymbol{v}) \leftarrow \frac{1}{K-1}\hat{v}'_K - \frac{K}{K-1}(\hat{v}_K)^2$
    if $\nu^2(\boldsymbol{v}) > \max(\nu^2(\boldsymbol{v}))$ then
        break
    $cVar \leftarrow -\infty$, $\bar{k} \leftarrow 0$, and $k \leftarrow 1$
    repeat
        $\hat{v}'_{K-1} \leftarrow \hat{v}'_K - v_k^2$ and $\hat{v}_{K-1} \leftarrow \hat{v}_K - \frac{v_k}{K}$
        $\nu^2(\boldsymbol{v}\backslash v_k) \leftarrow \frac{1}{K-2}\hat{v}_{K-1} - \frac{K-1}{K-2}(\hat{v}_{K-1})^2$
        if $\nu^2(\boldsymbol{v}\backslash v_k) < cVar$ then
            $cVar \leftarrow \nu^2(\boldsymbol{v}\backslash v_k)$
            $\bar{k} \leftarrow k$
        $k \leftarrow k + 1$
    until $k \leq K$
    Mark $v_{\bar{k}}$ as erroneous and delete
    if $\bar{k} \neq 1$ and $\bar{k} \neq K$ then
        Use $v_{\bar{k}-1}$ and $v_{\bar{k}+1}$ to interpolate the value of $v_{\bar{k}}$
    else
        Extrapolate $v_{\bar{k}}$
```

4.6 Probabilistic View Generation

Recall Definition 4.2 that defines the query for generating probability values for a tuple-independent probabilistic database (view). To precisely specify the user-defined range Ω in the definition, we define $\Omega = \{\hat{s}_l + u\Delta | u = -\frac{U}{2}, \ldots, \frac{U}{2}\}$, where Δ is a positive real number and U is an even integer. We refer to Δ and U as *view parameters.* These parameters describe U ranges of size Δ around the expected true value $\hat{s}_l$. In the online mode of our system, the query is evaluated at each time when a new value is streamed to the system. In the offline mode, all necessary parameters can be specified by users using a SQL-like syntax. For example, the syntax in Query 4.1 creates the probabilistic view in Figure 4.2.

```
CREATE VIEW prob_view AS DENSITY s OVER l OMEGA delta=2, U=2
FROM raw_values WHERE l >= 1 AND l <= 3
```

Query 4.1: Example of the probabilistic view generation query.

In the example shown in Query 4.1, `AS DENSITY s OVER t` illustrates the time-varying density for time series `s`. The `OMEGA` clause specifies the ranges of the data values of the probabilistic view, and the `WHERE` clause defines a time interval. Notice that the query given in Definition 4.2 is evaluated at each time l to obtain Λ_l. Specifically, at each l and for each $u = \{-\frac{U}{2}, \ldots, (\frac{U}{2} - 1)\}$ we compute the following integral:

$$\begin{aligned} o_u &= \int_{\hat{s}_l + u\Delta}^{\hat{s}_l + (u+1)\Delta} \mathbb{P}_l(R_l) dR_l, \\ &= \mathrm{P}_l(R_l = \hat{s}_l + (u+1)\Delta) - \mathrm{P}_l(R_l = \hat{s}_l + u\Delta), \end{aligned} \tag{4.9}$$

where $\mathrm{P}_l(R_l)$ is the cumulative distribution function of s_l.

In short, Eq. (4.9) involves computing $\mathrm{P}_l(R_l)$ for each value of $u = \{-\frac{U}{2}, \ldots, \frac{U}{2}\}$. Unfortunately, this computation may incur high cost when the time interval specified by the query spans over many days comprising of a large number of raw values. Moreover, this processing becomes significantly challenging when the query requests for a view with finer granularity (low Δ) and large range U, since such values for the view parameters considerably increase the computational cost.

To address this problem, we propose an approach that caches and reuses the computations of $\mathrm{P}_l(R_l)$, which were already performed at earlier times. The intuition behind this approach is to observe that probability distributions for a time series do not generally exhibit dramatic changes in short terms. For example, temperature values often exhibit only slight changes within short time intervals. In addition, similar probability distributions may be found periodically (e.g., early morning hours every day). Thus, the query processing can take advantage of the results from previous computation. In the rest of this section, we introduce an effective caching mechanism, termed *σ–cache*, that substantially boosts the performance of query evaluation by caching the values of $\mathrm{P}_l(R_l)$.

4.6.1 σ–cache

As introduced before, let $\mathrm{P}_l(R_l)$ be a Gaussian cumulative distribution function of s_l at time l. If required for clarity, we denote it as $\mathrm{P}_l(R_l; \hat{\theta}_l)$ where $\hat{\theta}_l = (\hat{s}_l, \hat{\sigma}_l^2)$. Observe that the shape of $\mathrm{P}_l(R_l; \hat{\theta}_l)$ is completely determined by $\hat{\sigma}_l^2$, since $\hat{s}_l$ only specifies the location of the curve traced by $\mathrm{P}_l(R_l; \hat{\theta}_l)$. This observation leads to an important property: suppose we move from time l to l', then the values of $\mathrm{P}_l(R_l = \hat{s}_l + u\Delta; \hat{\theta}_l)$, $\mathrm{P}_{l'}(R_{l'} = \hat{s}_{l'} + u\Delta; \hat{\theta}_{l'})$, and consequently o_u are the same if $\hat{\sigma}_l$ is equal to $\hat{\sigma}_{l'}$. We illustrate this property graphically in Figure 4.7. Moreover, since the shapes of $\mathrm{P}_l(R_l; \hat{\theta}_l)$ and $\mathrm{P}_{l'}(R_{l'}; \hat{\theta}_{l'})$ solely depend on $\hat{\sigma}_l$ and $\hat{\sigma}_{l'}$ respectively, we can assume in the rest of the analysis that the mean values of $\mathrm{P}_l(R_l)$ and $\mathrm{P}_{l'}(R_{l'})$ are zero. This could be done using a simple mean shift operation on $\mathrm{P}_l(R_l)$ and $\mathrm{P}_{l'}(R_{l'})$.

Our aim is to approximate $\mathrm{P}_{l'}(R_{l'})$ with $\mathrm{P}_l(R_l)$. This is possible only if we know how the distance (similarity) between $\mathrm{P}_l(R_l; \hat{\theta}_l)$ and $\mathrm{P}_{l'}(R_{l'}; \hat{\theta}_{l'})$ behaves as a function of $\hat{\sigma}_l$ and $\hat{\sigma}_{l'}$. If we know this relation then we can, with a certain error, approximate $\mathrm{P}_{l'}(R_{l'}; \hat{\theta}_{l'})$ with $\mathrm{P}_l(R_l; \hat{\theta}_l)$ simply by looking up $\hat{\sigma}_l$ and $\hat{\sigma}_{l'}$. Thus, if we have already computed $\mathrm{P}_l(R_l; \hat{\theta}_l)$ at time l then we can reuse it at time l' to approximate $\mathrm{P}_{l'}(R_{l'}; \hat{\theta}_{l'})$.

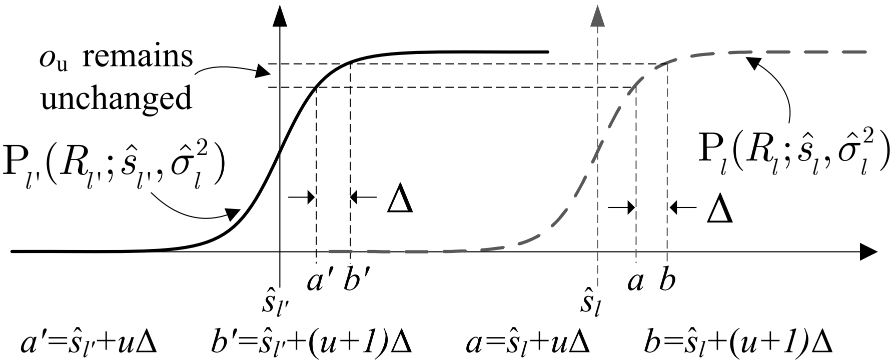

o_u remains unchanged
$P_{l'}(R_{l'};\hat{s}_{l'},\hat{\sigma}_l^2)$
Δ
Δ
$P_l(R_l;\hat{s}_l,\hat{\sigma}_l^2)$
a' b'
$\hat{s}_{l'}$
$\hat{s}_l$
a b
$a'=\hat{s}_{l'}+u\Delta$
$b'=\hat{s}_{l'}+(u+1)\Delta$
$a=\hat{s}_l+u\Delta$
$b=\hat{s}_l+(u+1)\Delta$

Figure 4.7: An example illustrating that o_u remains unchanged under mean shift operations when two Gaussian distributions have equal variance.

4.6.2 Constraint-Aware Caching

In practice, systems that use the σ–cache could have constraints of limited storage size or of error tolerance. To reflect this, we guarantee certain user-defined constraints. Specifically, we focus on the following:

- *Distance constraint* guarantees that the maximum approximation error is upper bounded by the distance constraint when the cache is used.
- *Memory constraint* guarantees that the cache does not use more memory than that specified by the memory constraint.

Before proceeding further, we first characterize the distance between two probability distributions using a measure known as the *Hellinger distance* [106]. It is a distance measure similar to the popular Kullback-Leibler divergence. However, unlike the Kullback-Leibler divergence, the Hellinger distance takes values between zero and one which makes its choice simple and intuitive. Formally, the square of Hellinger distance H between $\mathrm{P}_l(R_l)$ and $\mathrm{P}_{l'}(R_{l'})$ is given as:

$$\mathsf{H}^2[\mathrm{P}_l(R_l),\mathrm{P}_{l'}(R_{l'})] = 1 - \sqrt{\frac{2\hat{\sigma}_l\hat{\sigma}_{l'}}{\hat{\sigma}_l^2+\hat{\sigma}_{l'}^2}}. \tag{4.10}$$

The Hellinger distance assigns minimum value of zero when $\mathrm{P}_{l'}(R_{l'})$ and $\mathrm{P}_l(R_l)$ are the same and vice versa.

Guaranteeing Distance Constraint.
We use the Hellinger distance to prove the following theorem that allows us to approximate $\mathrm{P}_{l'}(R_{l'})$ with $\mathrm{P}_l(R_l)$.

THEOREM 4.1: Given $\mathrm{P}_{l'}(R_{l'})$, $\mathrm{P}_l(R_l)$, and a user-defined distance constraint H', we can approximate $\mathrm{P}_{l'}(R_{l'})$ with $\mathrm{P}_l(R_l)$, such that $\mathsf{H}[\mathrm{P}_l(R_l),\mathrm{P}_{l'}(R_{l'})] \leq \mathsf{H}'$, where $\hat{\sigma}_{l'} = \pi_s \cdot \hat{\sigma}_l$ and $\hat{\sigma}_{l'} > \hat{\sigma}_l$. The parameter π_r can be chosen as any value satisfying,

$$\pi_r \leq \frac{2+\sqrt{4-4\left(1-\mathsf{H}'^2\right)^4}}{2\left(1-\mathsf{H}'^2\right)^2}. \tag{4.11}$$

PROOF. Substituting $\hat{\sigma}_{l'} = \pi_r \cdot \hat{\sigma}_l$ in Eq. (4.10) we obtain,

$$(1 - \mathsf{H}'^2)\sqrt{1 + \pi_r^2} - \sqrt{2 \cdot \pi_r} = 0.$$

Solving for π_r we obtain,

$$\pi_r \leq \frac{2 + \sqrt{4 - 4\left(1 - \mathsf{H}'^2\right)^4}}{2\left(1 - \mathsf{H}'^2\right)^2}.$$

Since π_r is monotonically increasing in H', choosing a value of π_r as given by the above inequality guarantees the distance constraint H'. □

The above theorem states that if we have a user-defined *distance constraint* H' then we can approximate $\mathrm{P}_{l'}(R_{l'})$ by $\mathrm{P}_l(R_l)$ only if $\hat{\sigma}_{l'} > \hat{\sigma}_l$ and π_r is chosen using Eq. (4.11). Moreover, since π_r is defined as the ratio between $\hat{\sigma}_{l'}$ and $\hat{\sigma}_l$ we call it the *ratio threshold.*

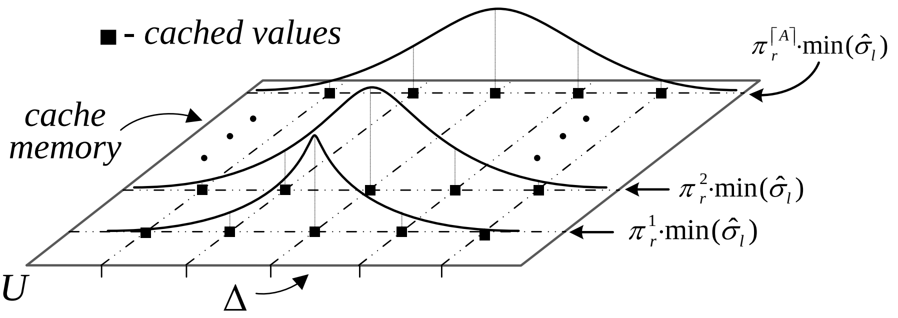

■ - cached values
cache memory
$\pi_r^{\lceil A \rceil} \cdot \min(\hat{\sigma}_l)$
$\pi_r^2 \cdot \min(\hat{\sigma}_l)$
$\pi_r^1 \cdot \min(\hat{\sigma}_l)$
U
Δ

Figure 4.8: Structure of the σ–cache.

Now, we describe how Theorem 4.1 allows us to efficiently store and reuse values of $\mathrm{P}_l(R_l)$ while query processing. First, we compute the maximum and minimum values amongst all $\hat{\sigma}_l$ matching the `WHERE` clause of the probabilistic view generation query (see Query 4.1). Let us denote these extremes as $\max(\hat{\sigma}_l)$ and $\min(\hat{\sigma}_l)$. We then define the *maximum ratio threshold* $\max(\pi_r)$ as,

$$\max(\pi_r) = \frac{\max(\hat{\sigma}_l)}{\min(\hat{\sigma}_l)}. \tag{4.12}$$

Given the user-defined distance constraint H' we use Eq. (4.11) to obtain a suitable value for π_r. Then we compute a A, such that,

$$\max(\hat{\sigma}_l) = \pi_r^A \cdot \min(\hat{\sigma}_l). \tag{4.13}$$

Let $\lceil x \rceil$ denote the smallest integer value that is not smaller than x. Then, $\lceil A \rceil$ gives us the maximum number of distributions that we should cache, such that the distance constraint is satisfied. We populate the cache by pre-computing values for $\lceil A \rceil$ distributions having standard deviations $\pi_r^a \cdot \min(\hat{\sigma}_l)$, where $a = 1, 2, \ldots, \lceil A \rceil$. As shown in Figure 4.8, these values are computed at points specified by the view parameters Δ and U.

We store each of these pre-computed distributions in a sorted container like a B-tree along with key $\pi_r^a \cdot \min(\hat{\sigma}_l)$. When we need to compute $\mathrm{P}_{l'}(R_{l'}; \hat{\theta}_{l'})$, we first look up the container to find keys $\pi_r^a \cdot \min(\hat{\sigma}_l)$ and $\pi_r^{a+1} \cdot \min(\hat{\sigma}_l)$, such that $\hat{\sigma}_{l'}$ lies between them. We then use the values associated with key $\pi_r^a \cdot \min(\hat{\sigma}_l)$ for approximating $\mathrm{P}_{l'}(R_{l'})$. By following this procedure we always guarantee that the distance constraint is satisfied due to Theorem 4.1.

Guaranteeing Memory Constraint.
Let us assume that we have a user-defined memory constraint M. We then consider an integer Q' which indicates the maximum number of distributions that can be stored in the memory size M. Here we prove an important theorem that enables the guarantee for memory constraint.

THEOREM 4.2: Given the values of Q', $\max(\hat{\sigma}_l)$, and $\min(\hat{\sigma}_l)$, the memory constraint M is satisfied if and only if the value of the ratio threshold π_r is chosen as,

$$\pi_r \geq \max(\pi_r)^{\frac{1}{Q'}}. \tag{4.14}$$

PROOF. From Eq. (4.13) we obtain,

$$\log_e(\max(\hat{\sigma}_l)) = Q' \cdot \log_e(\pi_r) + \log_e(\min(\hat{\sigma}_l)),$$
$$\pi_r = \max(\hat{\sigma}_l)^{\frac{1}{Q'}} \cdot \min(\hat{\sigma}_l)^{-\frac{1}{Q'}}.$$

From the above equation we can see that π_r is monotonically decreasing in Q'. Since $\max(\pi_r) = \frac{\max(\hat{\sigma}_l)}{\min(\hat{\sigma}_l)}$, we obtain,

$$\pi_r \geq \max(\pi_r)^{\frac{1}{Q'}}.$$

Choosing a value for π_r as given in the above equation guarantees that at most Q' distributions are stored, thus guaranteeing the memory constraint M. □

The above theorem states that given user-defined memory constraint Q' we set π_r according to Eq. (4.14) so as not to store more than Q' distributions. Also, given a distance constraint H' the rate at which the memory requirement grows is $\mathcal{O}(\log(\pi_r))$. Thus the cache size does not depend on the *number* of tuples that match the `WHERE` clause of the query in Query 4.1. Instead, it only grows logarithmically with the ratio between $\max(\hat{\sigma}_l)$ and $\min(\hat{\sigma}_l)$. Observe that the number of distributions stored by the σ–cache is independent from the view parameters Δ and U. This is a desirable property since it implies that, queries with finer granularity are answered by storing the same number of distributions.

There is an interesting trade-off between the distance constraint and the memory constraint (see Eq. (4.11) and Eq. (4.14)). When the distance constraint increases, the amount of memory required by the σ–cache decreases in order to guarantee the distance constraint and vice versa. Thus, as expected, there exists a give-and-take relationship between available memory size and prescribed error tolerance.

In the following section, we will demonstrate significant improvement with respect to query processing by using the σ–cache.

4.7 Experimental Evaluation

The main goals of our experimental study are fourfold. First, we show that the performance of the proposed dynamic density metrics, namely, ARMA-GARCH and Kalman-GARCH are efficient and accurate over real-world data. Second, we compare the performance of the ARMA-GARCH metric with that of the C-GARCH enhancement, in order to show that C-GARCH is efficient as well as accurate in handling erroneous values in time series. We then demonstrate that the use of the σ–cache significantly increases query processing performance. Lastly, we perform experiments validating that real world datasets exhibit regimes of changing volatility.

In our experiments, we use two real datasets, details of these datasets are as follows:

Campus Dataset: This dataset comprises of ambient temperature values recorded over twenty five days. It consists of approximately eighteen thousand samples. These values are obtained from a real sensor network deployment on the EPFL university campus in Lausanne, Switzerland. We refer to this dataset as *campus-data*.

Moving Object Dataset: This dataset consists of GPS logs recorded from on-board navigation systems in 192 cars in Copenhagen, Denmark. Each log entry consists of time and x-y coordinate values. In our evaluation we use only x-coordinate values. This dataset contains approximately ten thousand samples recorded over five and half hours. We refer to this dataset as *car-data*.

Table 4.2 provides a summary of important properties of both datasets. We have implemented all our methods using MATLAB Ver. 7.9 and Java Ver. 6.0. We use a Intel Dual Core 2 GHz machine having 3GB of main memory for performing the experiments.

Table 4.2: Summary of Datasets

	campus-data	*car-data*
Monitored parameter	Temperature	GPS Position
Number of data values	18031	10473
Sensor accuracy	± 0.3 deg. C	± 10 meters
Sampling interval	2 minutes	1-2 seconds

4.7.1 Comparison of Dynamic Density Metrics

We compare our main proposals (ARMA-GARCH and Kalman-GARCH) with uniform thresholding (UT) and variable thresholding (VT). These evaluations are performed on both datasets. As described in Section 4.2, we used the density distance for comparing the quality of distributions obtained using the dynamic density metrics.

Figure 4.9 shows a comparison of density distance for the various dynamic density metrics for both datasets along with increasing window size (H). Clearly, both the ARMA-GARCH metric and the Kalman-GARCH metric outperform the naive density metrics. Specifically, those advanced dynamic density metrics outperform the naive

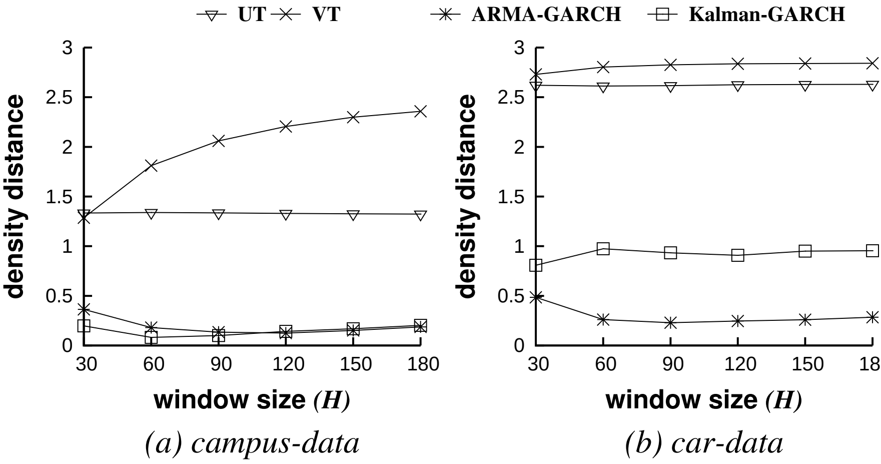


Figure 4.9: Comparing quality of the dynamic density metrics.

density metrics by giving *upto 20 times and 12.3 times lower* density distances for *campus-data* and *car-data* respectively.

Among the advanced dynamic density metrics, the ARMA-GARCH metric performs better than all the other metrics. For *car-data* we can observe that the Kalman-GARCH metric gives low accuracy as the window size increases. This behavior is expected since when larger window sizes are used for the Kalman Filter, there is a greater chance of error in inferring $\hat{s}_l$. In our observation, the use of smaller window sizes (e.g., $H = 10$) for the Kalman-GARCH metric performs twice better, compared to the ARMA-GARCH metric.

Next, we compare the efficiency of the dynamic density metrics. Figure 4.10 shows the average times required to perform one iteration of density inference. Because of the large performance gain of the ARMA-GARCH metric, the execution times are shown on logarithmic scale. The ARMA-GARCH metric achieves a *factor of 5.1 to 18.6 speedup* over the Kalman-GARCH metric. This is due to slow convergence of the iterative EM

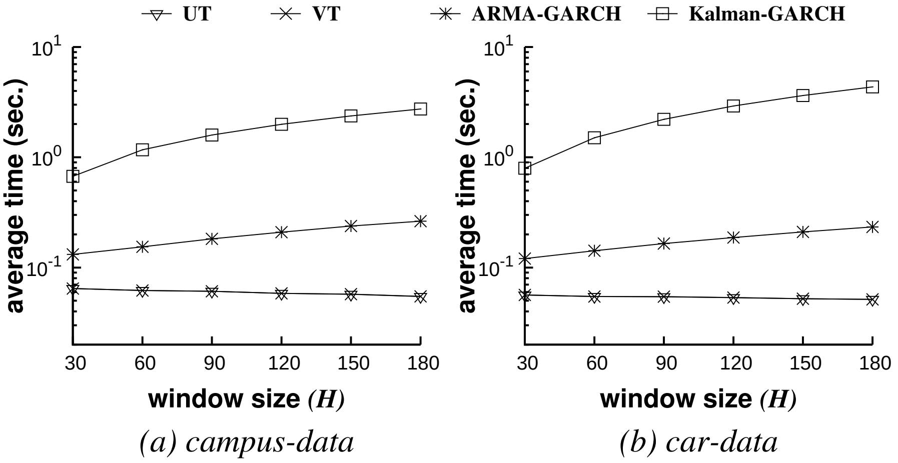


Figure 4.10: Comparing efficiency of the dynamic density metrics. Note the logarithmic scale on the y-axis.

(Expectation-Maximization) algorithm used for estimating parameters of the Kalman Filter. Thus, unlike the ARMA model, computing parameters for the Kalman Filter takes longer for large window sizes. The naive dynamic density metrics are much more efficient than the Kalman-GARCH metric. But they are only marginally better than the ARMA-GARCH metric. Overall the ARMA-GARCH metric shows excellent characteristics in terms of both efficiency and accuracy.

In the next set of experiments, we discuss the effect of model order of an ARMA(α,0) model on density distance. Figure 4.11 shows the density distance obtained by using several metrics when the model order α increases. Observe that for the ARMA-GARCH metric the density distance increases with model order. This justifies our choice of a low model order for the ARMA-GARCH metric.

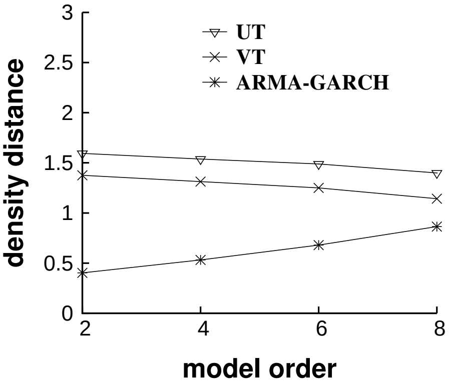
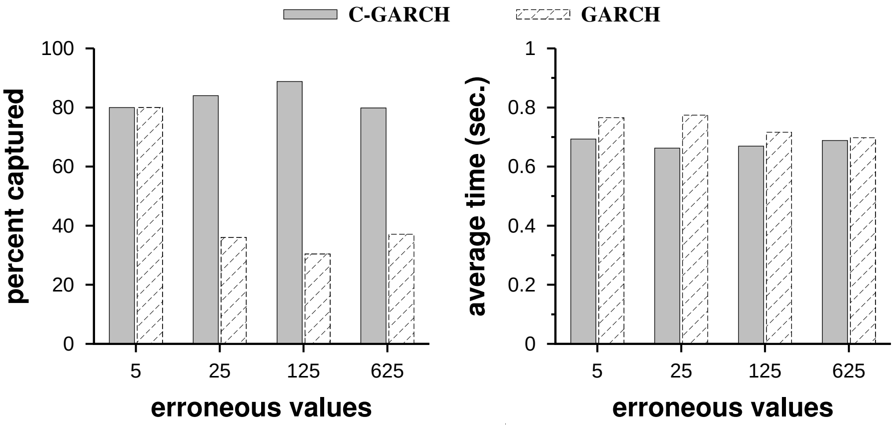


Figure 4.11: Effect of model order on *campus-data*.

4.7.2 Impact of C-GARCH

In the following, we demonstrate the improved performance of the C-GARCH model by comparing it with the plain ARMA-GARCH metric using *campus-data* (we omit the results from *car-data* because they are similar). We start by inserting erroneous values synthetically, since for comparing accuracy we should know beforehand the number of erroneous values present in the data. The insertion procedure inserts a pre-specified number of very high (or very low) values uniformly at random in the data.

For evaluating the C-GARCH approach we first compute $\max(\nu^2(\boldsymbol{v}))$ using a given set of clean values and then execute the C-GARCH model while setting $\varsigma = 8$. Figure 4.12 (a) compares the percentage of total erroneous values detected for C-GARCH and ARMA-GARCH. Admittedly, the C-GARCH approach is *more than twice effective* in detecting and cleaning erroneous values. Additionally, from Figure 4.12 (b) it can be observed that the C-GARCH approach does not require excessive computational cost as compared to ARMA-GARCH. The reason is that the ARMA model estimation takes more time if there are erroneous values in the window $\boldsymbol{s}_{t-1}^{H}$. This additional time offsets the time spent by the C-GARCH model in cleaning erroneous values before they are given to the ARMA-GARCH metric.



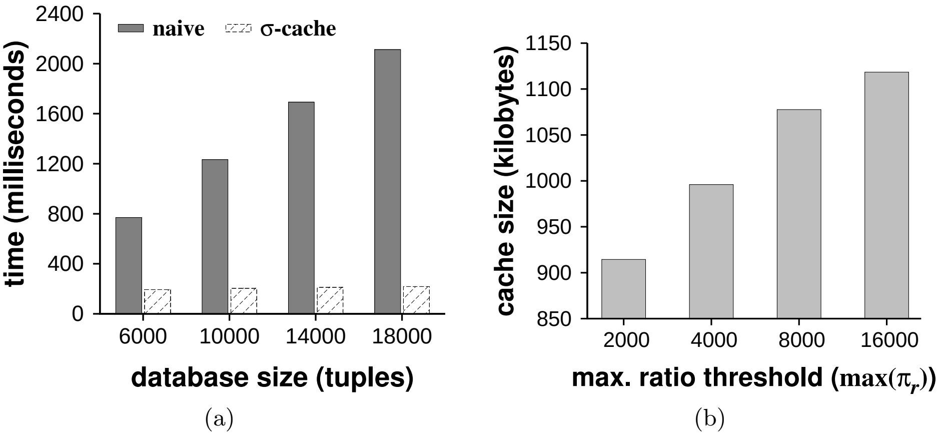
(a) *(b)*

Figure 4.12: Comparing C-GARCH and GARCH. (a) Percentage of erroneous values successfully detected and (b) average time for processing a single value.

4.7.3 Impact of using σ–cache

Next, we show the impact of using the σ–cache while creating a probabilistic database. Particularly, we are interested in knowing the increase in efficiency obtained from using a σ–cache. Moreover, we are also interested in verifying the rate at which the size of the σ–cache grows as the maximum ratio threshold $\max(\pi_r)$ increases. Here, we expect the cache size to grow logarithmically in $\max(\pi_r)$.

We use *campus-data* for demonstrating the space and time efficiency of the σ–cache. We choose $\Delta = 0.05$, $U = 300$, Hellinger distance $\mathsf{H} = 0.01$, and compute π_r using Eq. (4.11). Figure 4.13 (*a*) shows the improvement in efficiency obtained for the probabilistic view generation query with increasing number of tuples. Here, the naive approach signifies that the σ–cache is not used for storing and reusing previous computation. In Figure 4.13 all values are computed by taking an ensemble average over ten independent



(a) (b)

Figure 4.13: (a) Impact of using the σ–cache on efficiency. (b) Scaling behavior of the σ–cache. Note the exponential scale on the x-axis.

executions. Clearly, using the σ–cache exhibits manyfold improvements in efficiency. For example, when there are 18K raw value tuples we observe a *factor of 9.6 speedup* over the naive approach. Figure 4.13 (b) shows the memory consumed by the σ–cache as $\max(\pi_r)$ is increased. As expected, the cache size grows only logarithmically as the maximum ratio threshold $\max(\pi_r)$ increases. This proves that the σ-cache is a space- and time-efficient method for seamlessly caching and reusing computation.

4.7.4 Verifying Time-Varying Volatility

Before we infer time-varying volatility using the ARMA-GARCH metric or the Kalman-GARCH metric it is important to verify whether a given time series exhibits changes in volatility over time. For testing this we use a null hypothesis test proposed in [120]. The null hypothesis tests whether the errors obtained from using a ARMA model (e_i^2) are independent and identically distributed ($i.i.d$). This is equivalent to testing whether $\Phi_1 = \cdots = \Phi_\zeta = 0$ in the linear regression,

$$e_i^2 = \Phi_0 + \Phi_1 e_{i-1}^2 + \cdots + \Phi_\zeta e_{i-\zeta}^2 + \epsilon_i, \tag{4.15}$$

where $i \in \{\zeta + 1, \ldots, H\}$, ϵ_i denotes the error term, $\zeta \geq 1$, and H is the window size. If we reject the null hypothesis (i.e., $\Phi_j \neq 0$) then we can say that the errors are not $i.i.d$, thus establishing that the given time series exhibits time-varying volatility. First, we start by computing the sample variance of e_i^2 and ϵ_i denoted as $\nu^2(e_i^2)$ and $\nu^2(\epsilon_i)$ respectively. Then,

$$\Xi(\zeta) = \frac{(\nu^2(e_i^2) - \nu^2(\epsilon_i))/\zeta}{\nu^2(\epsilon_i)/(H - 2\zeta - 1)}, \tag{4.16}$$

is asymptotically distributed as a chi-square distribution χ_ζ^2 with ζ degrees of freedom. Thus we reject the null hypothesis if $\Xi(\zeta) > \chi_\zeta^2(0.05)$, where $\chi_\zeta^2(0.05)$ is in the upper $100(1 - 0.05)^{th}$ or 95^{th} percentile of χ_ζ^2 or the p-value of $\Xi(\zeta) < 0.05$ [120].

To show that our datasets exhibit regimes of changing volatility we compute the value of $\Xi(\zeta)$ where $\zeta = \{1, 2, \ldots, 8\}$ on 1800 windows containing 180 samples each (i.e., $H = 180$) for *campus-data* and *car-data*. Then we reject the null hypothesis if the average value of $\Xi(\zeta)$ over all windows is greater than $\chi_\zeta^2(0.05)$.

Figure 4.14 shows the results from this evaluation. Clearly, we can reject the null hypothesis for both datasets because for all values of ζ, $\chi_\zeta^2(0.05)$ is much lower than $\Xi(\zeta)$. This means that e_i^2 are not $i.i.d$ and thus we can find regimes of changing volatility. Interestingly, for *car-data* (see Figure 4.14 (b)) we can see that $\chi_\zeta^2(0.05)$ and $\Xi(\zeta)$ are close to each other. Thus the *car-data* contains less time-varying volatility as compared to the *campus-data*.

The above results support the claim that real datasets show change of volatility with time, thus justifying the use of the GARCH model.

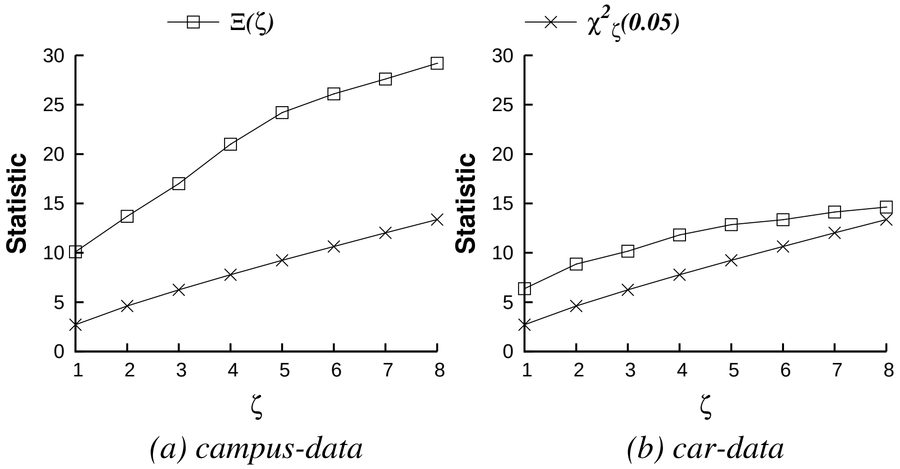


Figure 4.14: Verifying time-varying volatility.

4.8 Related Work

In order to effectively deal with uncertain data, a vast body of research on probabilistic databases has been conducted in the literature, including concepts and foundations [20, 35, 77], query processing [34, 72, 91, 100], and indexing schemes [67, 80, 124]. All these studies, however, share the common condition that probability values associated with data must be given a priori. As a result, a large variety of applications are still incapable of receiving benefits from such well-established tools for processing probabilistic databases, due to the lack of methods for establishing the required probability values.

Some previous work highlights the fact that creating probabilistic databases is a non-trivial problem. They then propose effective solutions for the problem; however, the studies have only limited scope for domain-specific applications, such as handling duplicated data records [10, 56] and building structured data from unstructured data [55]. series are still unable to benefit from the research on processing probabilistic databases.

More recently, the concept of probabilistic databases has been extended into stream data processing, so-called *probabilistic streams* [32, 80, 110]. associated with a probability that would mean as “Alice is at room A with a 30 % chance”. Ré *et al.* [110] propose a framework for query processing over probabilistic (Markovian) streams. Later, an access method for such Markovian streams is introduced in [80] for efficient query processing. Cormode and Garofalakis [32] also propose efficient algorithms based on hash-based sketch synopsis structure for processing aggregate queries over probabilistic streams. While all these studies assume probabilistic streams are given beforehand, Tran *et al.* [126] introduce a complete solution to create probabilistic streams. Unfortunately, this proposal is focused on RFID data, whereas our solution accepts arbitrary time-series data including such RFID data.

Processing probabilistic queries is another related area to our work. Cheng *et al.* [24] introduce several important types of probabilistic queries, as well as a generic query evaluation framework over inherently imprecise data. Although they assume that an uncertainty bound for data can be easily given by users, the assumption may not hold in many real-world applications. Deshpande and Madden [40] introduce the abstraction of *model-based views* that are database views created from the underlying data by applying numerical models. These views are then used for query processing instead of using the actual data. This idea is then extended by Kanagal and Deshpande [66], in which various particle filters are used for generating model-based views. This proposal requires a sufficient number of generated particles to obtain reliable probabilistic inferences, however, this substantially decreases the efficiency of the system.

Some prior research focuses on system perspectives associated with uncertain data. Wang *et al.* [128] introduce Bayesstore which stores joint probability distribution functions encoded in a Bayesian network. Jampani *et al.* [61] propose a novel concept, by which the system does not store probabilities but parameters for generating the probabilities. Our work inherits this idea. Antova *et al.* [11] introduce the abstractions of world-sets and world-tables for capturing attribute-level uncertainty and possible world semantics of a probabilistic database. Cheng *et al.* [26] propose U-DBMS for managing uncertain data where the probability density function for the uncertain attributes is pre-specified.

4.9 Conclusion

In this chapter, we proposed a novel and generic solution for creating probabilistic databases from imprecise time-series data. Our solution includes two novel components: the dynamic density metrics that effectively infer time-dependent probability distributions for time series and the Ω–View builder that uses the inferred distributions for creating probabilistic databases. We also introduced the σ–cache that enables efficient creation of probabilistic databases while obeying user-defined constraints. We demonstrated that by using the σ–cache the efficiency of creating probabilistic databases can be enhanced by upto an order of magnitude. Many other comprehensive experiments highlight the effectiveness of our approaches. In the next chapter, we shall consider the problems arising while managing and querying community-sensed data.

Appendix 4.A Probabilistic Query Evaluation

Cheng *et al.* [24] defined queries for processing uncertain data. One of the assumptions in their work was that the area of uncertainty of a particular raw value is finite. Thus, the probability distributions for raw values were defined over a finite interval. On the contrary, in our proposal raw values follow a Gaussian distribution. This makes query evaluation more challenging since the Gaussian distribution is not defined over a finite interval. As we shall see, it is possible to evaluate most of the queries by using closed form expressions and, more importantly, without imposing a finite area of uncertainty.

Probabilistic queries on uncertain data are classified as: (a) *value-based queries* that return a single number or aggregate, (b) *entity-based queries*, that return a set of objects. These broad query types are further classified as:

- Aggregate Value-Based Queries: VAvgQ, VSumQ, VMinQ, and VMaxQ
- Aggregate Entity-Based Queries : EMinQ and EMaxQ.

All the queries listed above take as input a set of probability distributions. This set is defined as $\mathcal{K} = \{\mathbb{P}_1(R_1), \mathbb{P}_2(R_2), \ldots, \mathbb{P}_n(R_n)\}$. In our case, we have $\mathbb{P}_j(R_j) \sim \mathcal{N}(\mu_j, \sigma_j^2)$ for all $1 \leq j \leq n$.

Definition 4.4: **VSumQ (VAvgQ) query.** Given probability distributions $\mathcal{K}$ the VSumQ (VAvgQ) query returns a probability distribution $\mathbb{P}_s(R_s)$ ($\mathbb{P}_a(R_a)$) where R_s (R_a) is a random variable for the sum (average) of $(R_1, R_2, \ldots, R_n)$.

This query could be easily answered since $\mathbb{P}_j(R_j)$ follows a Gaussian distribution. One nice property of Gaussian distributions is that the sum and average of Gaussian random variables also follows a Gaussian distribution. Thus, sum and average of $(R_1, R_2, \ldots, R_l)$ also obeys a Gaussian distribution. Moreover, $\mathbb{P}_s(R_s) \sim \mathcal{N}(\mu_s, \sigma_s^2)$ where $\sigma_s^2 = \sum_{j=1}^{n} \sigma_j^2$ and $\mu_s = \sum_{j=1}^{n} \mu_j$. Similarly, $\mathbb{P}_a(R_a) \sim \mathcal{N}(\mu_a, \sigma_a^2)$ where $\sigma_a^2 = \frac{1}{n^2} \sum_{j=1}^{n} \sigma_j^2$ and $\mu_a = \frac{1}{n} \sum_{j=1}^{n} \mu_j$. Note that this assumes that $R_1, \ldots, R_n$ are independent.

Definition 4.5: **VMinQ (VMaxQ) query.** Given probability distributions $\mathcal{K}$ the VMinQ (VMaxQ) query returns a probability distribution $\mathbb{P}_{min}(R_{min})$ ($\mathbb{P}_{max}(R_{max})$) where $R_{min}(R_{max})$ is a random variable for the minimum (maximum) of $(R_1, R_2, \ldots, R_n)$.

The VMinQ (VMaxQ) could be answered using extreme value distributions. Particularly, we use the Gumbel distribution to model the maximum of a set of Gaussian random variables,

$$\mathbb{P}_{max}(R_{max} = r) = \frac{1}{\theta_1} \left(e^{-\frac{r-\theta_2}{\theta_1}} \right) \left(e^{-e^{-\frac{r-\theta_2}{\theta_1}}} \right), \tag{4.17}$$

here parameters θ_1 and θ_2 could be estimated from data. The distribution for R_{min} can be obtained by replacing r with $-r$ in Eq. (4.17).

The next type of queries we discuss are the aggregate-based entity queries (EMinQ and EMaxQ). These queries typically assume that each probability distribution from $\mathcal{K}$ is associated with a time-series data source (for example, sensor, transmitter, GPS device, etc.). Thus, let use assume that we have $\mathcal{W} = \{w_1, w_2, \ldots, w_n\}$ data sources where each w_j is associated with a probability distribution $\mathbb{P}_j(R_j)$ from $\mathcal{K}$.

Definition 4.6: **EMinQ (EMaxQ) query.** Given the data sources $\mathcal{W}$ and their corresponding probability distributions $\mathcal{K}$ the EMinQ (EMaxQ) query returns a set of tuples $\mathcal{O} = \{(w_1, o_1), (w_2, o_2), \ldots, (w_n, o_n)\}$ where o_j is the probability that R_j is the minimum (maximum) amongst all entities $\mathcal{W}$, where $1 \leq j \leq n$.

Now, o_j can be derived as,

$$o_j = \int_{-\infty}^{+\infty} \mathbb{P}_j(R_j = u) \prod_{k=1 \wedge k \neq j}^{n} (1 - \mathrm{P}_k(R_k = u))du, \tag{4.18}$$

where $\prod_{k=1 \wedge k \neq j}^{n}(1 - \mathrm{P}_k(R_k = u))$ is the probability that the values of all data sources except w_j is greater than u. Thus Eq. (4.18) gives us the probability that w_j has the minimum value. The expression for w_j in Eq. (4.18) is a complicated integral which is difficult to simplify. Therefore, we propose using a Monte Carlo method for evaluating the integral in Eq. (4.18). Observe that Eq. (4.18) can be interpreted as,

$$o_j = \mathrm{E}_j \left[\prod_{k=1 \wedge k \neq j}^{n} (1 - \mathrm{P}_k(R_k = x)) \right], \tag{4.19}$$

where E_j is the expectation *w.r.t.* $\mathbb{P}_j(R_j)$. Thus, if we draw L random variates according to $\mathbb{P}_j(R_j)$ as $(u_1, \ldots, u_L)$, then Eq. (4.19) can be evaluated as,

$$o_j = \frac{1}{L} \sum_{i=1}^{L} \prod_{k=1 \wedge k \neq j}^{n} (1 - \mathrm{P}_k(R_k = u_i)). \tag{4.20}$$

This completes our discussion on evaluating probabilistic queries. We have shown that aggregate value-based queries (VMinQ, VMaxQ, VSumQ, VAvgQ) could be evaluated using closed form expressions and aggregate entity-based queries (EMinQ and EMaxQ) can be evaluated using Monte Carlo integration. Particularly, we have demonstrated that probabilistic queries can be evaluated when the region of uncertainty is not finite.

Chapter 5

ConDense: Managing Data in Community-Driven Mobile Geosensor Networks

> *It is better to be vaguely right than exactly wrong.*
>
> Carveth Read, 1914

5.1 Introduction

In this chapter, we propose methods for concisely modeling and managing data from community-driven sensor networks. Research in mobile geosensor networks is rapidly evolving to investigate the novel paradigm of community-driven sensing. In community-driven sensing, sensors of various sorts (e.g., multi-sensor units monitoring air quality, cell phones, thermal watches, thermometers in vehicles, etc.) are carried by the community (public vehicles, private vehicles, or individuals) during their daily activities, collecting data about the environment.

At its core, community sensing is a new form of mobile geosensor network [5]. Unique characteristics of this sensing paradigm lie in its organic and unstructured mobile sensing. This is analogous to the Web 2.0 model, where the community participates in generating data. This differs from traditional mobile geosensor networks, where the primary objective is to monitor the environment through a controlled specification of desired sampling, mobility characteristics, or through appropriate sensor placement [85, 98].

This chapter investigates different approaches of *condensing*[1] the data generated by large-scale Community-driven Mobile GeoSensor Networks (CGSN). We present ConDense (Community-driven Sensing of the Environment), a framework for efficiently managing data generated about the environment. The ConDense framework takes into

---

[1] con·dens·ing (v.intr.): To make more concise; abridge or shorten.

account the unique properties of CGNSs and treats the underlying sensor network as a disconnected component, which is collecting data using local policies and principles. Although there is significant literature on model-based query processing on mobile sensor networks, there is a lack of understanding of approaches to determine high quality and concise models of the phenomenon from CGSNs. The models built using the raw data are necessary, since raw data generated from sensors is often, *"imprecise and erroneous, hence rarely usable as it is"* [40]. The raw data generated needs to be synthesized and managed for consumption by scientists, applications, and the community.

Regression-based modeling approaches have been proposed in the literature to provide mathematically meaningful descriptions of the sensed phenomenon. For example, [40] presents such a model-based view of sensory readings (temperature in rooms). Here, the applications query only the models, and the models, in turn, get updated as time progresses and new data arrives. However, most prior work implicitly assumes that the sensors are relatively homogeneously distributed and/or their sensing behavior can be tuned, considering the phenomenon being sensed. Typically, trials have used small-scale deployments (e.g., covering a room or a small field).

Unfortunately, CGSNs cannot be tightly controlled and deployments cover large areas (e.g., part of a city or state). Hence, it is difficult to produce a homogeneous, good quality view of the phenomenon. The community-sensing pattern leads to spatio-temporal irregularities in the sensing; while some areas might be adequately sampled, some other areas would not be. A challenging question is: *how do we efficiently create quality-controlled models that cover the sensed data, spatially and temporally?*

Traditional geo-statistical techniques, like Kriging, [28] can be used for modeling such phenomenon. Kriging interpolates the best linear unbiased estimate of a value at an unobserved point in space, based on the weighted linear combination of surrounding observations, minimizing the approximation error. We found, however, such approaches incur high computational complexity, and hence suffer from scaling issues with dynamic temporal variations. On the other extreme, a naïve strategy would be to grid the area under consideration into equal size grid cells and compute a model per grid cell. This approach is simple, however, might lead to lower quality models.

5.1.1 Chapter Organization

We begin by describing the sensor deployments in Section 5.2. In Section 5.3 we survey the related work. We define the problem of concisely modeling community-sensed data in Section 5.4. We propose non-adaptive and adaptive solutions for this problem in Section 5.5 and Section 5.6 respectively. Our main proposal, the adaptive k-means algorithm, is described in Section 5.6.2. In Section 5.7, we perform an extensive experimental evaluation of our approaches on two real-world community-sensed datasets.

5.2 Sensors, Deployment, and Data Collection

For experiments and evaluation, we consider two sensor network deployments, namely OpenSense and Safecast. In this section, we discuss the details of the sensors, which are a part of these deployments, and the datasets that are collected for experimental evaluation.

Opensense: The OpenSense [5] project (the main source of funding for this work) currently has two deployments, in the cities of Lausanne and Zurich in Switzerland. In both deployments, the sensors are placed on public transport vehicles, like buses or trams, and additionally include stationary monitoring stations at strategic locations. Figure 5.1 shows the infrastructural overview of the OpenSense deployments. The sensors monitor the concentration of various environmental pollutants like, Carbon Monoxide (CO), Carbon Dioxide (CO_2), Nitrogen Dioxide (NO_2), and Ozone (O_3). Table 5.1 shows the important characteristics of the sensors used for monitoring these pollutants.

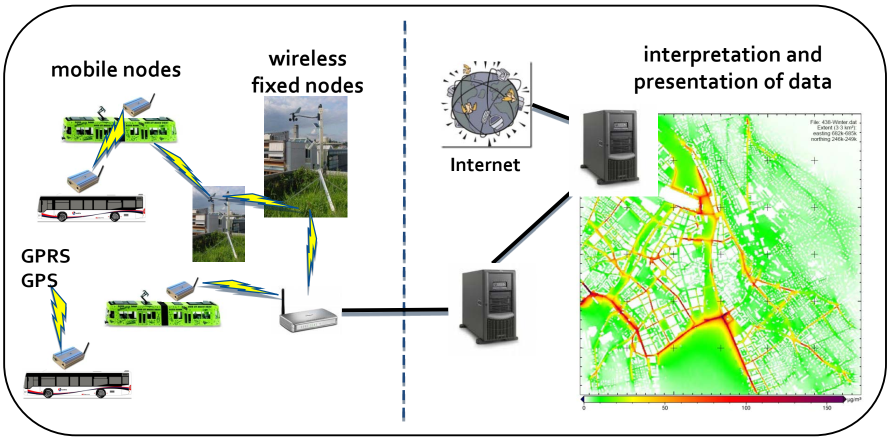


Figure 5.1: Community-driven mobile geosensor network infrastructure.

The normal urban concentration shown in Table 5.1 is the permissible concentration of a pollutant in an urban environment. These concentrations are given by the National Ambient Air Quality Standards (NAAQS) [1] based in the United States. As will be discussed in Section 5.4, these normal urban concentration ranges will be used for weighting the approximation errors made while approximating the pollutant concentration using a model.

Table 5.1: Characteristics of sensors and pollutants.

Pollutant	Type	Normal Urban Concentration	Average Power
NO_2	electrochemical	0.008 to 0.04 ppm	45 mW
CO	electrochemical	0.5 to 5 ppm	0.85 mW
CO_2	electrochemical	500 to 1500 ppm	0.5 W
O_3	semi-conductor	0.05 to 0.15 ppm	-
Radiation	event counter	0 to 0.23 μSv/h	-

We use the dataset collected from a mobile station mounted on a tram in Zurich, Switzerland. This dataset was collected over seven weeks. For our experimental evaluation, we use the Ozone (O_3) values. The sensors mounted on the tram follow a local sampling policy. An important property of this data is that it was collected from a relatively clean environment of Zurich, therefore this dataset does not contain large amount of variation in the values of O_3, NO_2, CO_2, etc. We denote this dataset as *opensense*.

Table 5.2: Summary of the Datasets.

	opensense	*safecast*
Monitored parameter	Ozone	Radiation Exposure
Number of data values	110,500	970,000
Sensor accuracy	±2 ppb	-*
Sampling interval	40 sec.	5 sec.

*Radiation counters have variable accuracy.

Safecast: The Safecast[2] project is a community-driven global sensor network deployment that was kick-started one week after the Fukushima Daiichi nuclear disaster[2] to monitor the radiation level in eastern Japan. The project enables people to both contribute and freely use the collected data. The project is a community-driven project with over one hundred volunteers contributing to the project.

The radiation data is collected by using: (a) 35 mobile stations that are attached to the cars of the volunteers, (b) 50 handheld stations, and (c) 50 static stations. The measurement unit of radiation is micro Sievert per hour (μSv/h). This unit evaluates the biological effects of radiation as opposed to other radiation units, which just measure the absorbed dose of radiation energy.

Since there are a variety of sensors being used for radiation measurements, the collected data is less accurate as compared to the OpenSense deployment. This dataset was collected over a period of twenty five weeks. We denote this dataset as *safecast*. Table 5.2 gives a summary of both the datasets.

5.3 Related Work

In environmental science, rich models are developed to model environmental phenomenon. For example, air quality models [4] consider three core aspects: pollution sources, transport (wind), and chemical processes. Models are built to predict expected pollution readings considering terrain characteristics, like, elevation, built-up areas, etc. Appropriate geo-statistical interpolation techniques like Kriging [28] or Gaussian plumes [92] are used to infer spatio-temporal models of the phenomenon. Validation is carried out using carefully designed sensor layouts, using few high-precision sensors.

While appropriate for visualization or creating rich models from the data, unfortunately, these geo-statistical techniques are unsuitable for modeling the CGSN data in

---

[2] `http://en.wikipedia.org/wiki/Fukushima_Daiichi_nuclear_disaster`

a database environment. This is because they take enormous computation time (e.g., of the order of hours [4]), and hence cannot be applied repeatedly to model error-prone and incomplete data streams from a geographical area. Database environments need to accept incoming sensory data and build models for consumption by queries. To do so, we need solutions that consider performance parameters like model quality, but also account for computational efficiency, query response time, down-time, etc.

In database environments, model-based approaches on distributed sensor data [14, 40, 53, 125] decouple the sensory updates from the query infrastructure by creating models of the underlying data and allowing the queries to view and operate on top of the models. There are different works that build temporal (per sensor) or spatial models on well-defined regions (for e.g., using grids [40]). Prior work has also suggested *in-network* modeling [14, 53] to reduce communication overhead.

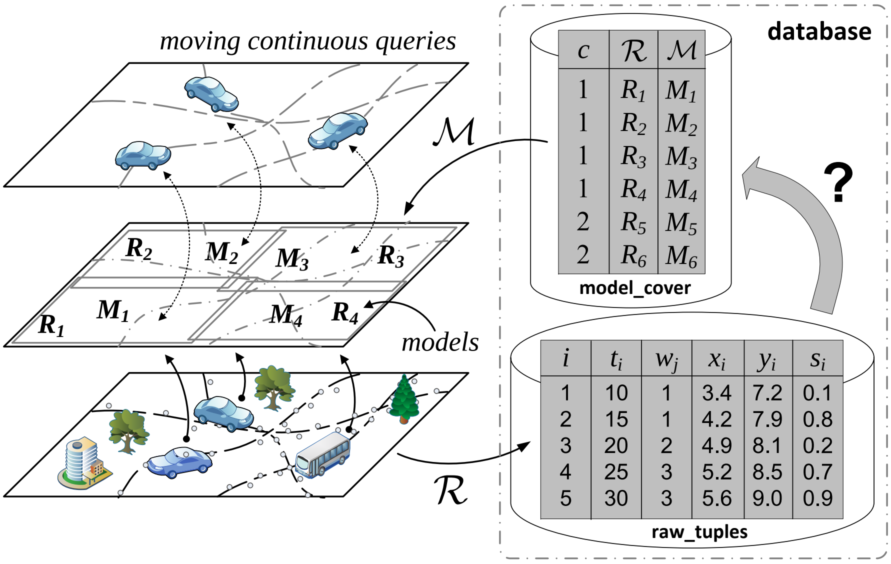
Such approaches have not considered the ramifications of developing models on top of the CGSN data. Firstly, unlike prior deployments, sensors in a CGSN have autonomous (buses) or uncontrolled (private cars) mobility. Hence per-sensor models are inappropriate, since the phenomenon changes behavior as the sensors move over larger areas like cities. Secondly, such approaches have problems with the quality of data. Prior approaches implicitly assume a quasi-uniform distribution of readings for learning the models (e.g., basis function selection or weight optimization). Community sensed data is unevenly distributed (skewed), spatially and temporally. Hence, it is challenging to design methods for quality-controlled model covers that have reasonable performance overhead.

As such, there are many projects today [3, 17, 33, 86, 130] exploring community-driven sensing of environmental phenomena. Most of these projects primarily focus on systems issues like developing inexpensive sensors, calibration, how to provide incentives to the users, reduce sampling overhead [76]. None of these projects investigate the research question of exploring efficient strategies to create a model-based data abstraction layer, suitable for database environments.

5.4 Problem Characterization

Before diving into the details, we present foundational definitions and establish the notation used in the rest of the chapter. We start by introducing the ConDense framework, which is shown as a schematic in Figure 5.2. For simplicity, we decompose this framework into the following three components:

Sensors: This component is responsible for sensing the environment. We assume that there are sensors that are moving over a geographical region $\mathcal{R}$ (refer Figure 5.2). For example, $\mathcal{R}$ could be a suburb, city, state, or even a larger geographical area. In addition, we consider sensors that are currently moving in the region $\mathcal{R}$ and are sensing the parameters of interest. In this chapter, we are interested in parameters like pollution and/or radiation.


moving continuous queries
database
$\mathcal{M}$
?
R_1 M_1 R_2 M_2 M_3 R_3 M_4 R_4
models
$\mathcal{R}$
model_cover
raw_tuples

c	$\mathcal{R}$	$\mathcal{M}$
1	R_1	M_1
1	R_2	M_2
1	R_3	M_3
1	R_4	M_4
2	R_5	M_5
2	R_6	M_6

i	t_i	w_j	x_i	y_i	s_i
1	10	1	3.4	7.2	0.1
2	15	1	4.2	7.9	0.8
3	20	2	4.9	8.1	0.2
4	25	3	5.2	8.5	0.7
5	30	3	5.6	9.0	0.9

Figure 5.2: Architecture of the ConDense framework.

We assume that the values transmitted by these sensors are continuously updated in a database table called `raw_tuples`. Each tuple i in the table of `raw_tuples` consists of the sensor identifier w_j, the time t_i at which the value was sensed, the GPS co-ordinates of the sensed value (x_i, y_i) and the sensor value s_i. Additionally, we denote a single raw tuple in the database as $b_i = (t_i, x_i, y_i, s_i)$, its position as $g_i = (x_i, y_i)$, and its positioned value as $v_i = (x_i, y_i, s_i)$.

Models: The modeling component provides a multi-model abstraction (i.e., model cover) over the raw tuples. On the one hand, it is responsible for answering continuous queries registered by the vehicles; and on the other hand, it is responsible for continuously maintaining the models that are obtained using raw tuples.

Our main objective in this chapter is to build and continuously maintain a model cover over the region $\mathcal{R}$. Before proceeding further, let us rigorously define a model cover.

DEFINITION 5.1: **Model Cover.** A *model cover* is defined as a set of models $\mathcal{M} = \{M_o | 1 \leq o \leq O\}$, where model M_o models the region $R_o \subseteq \mathcal{R}$ respectively, for all $1 \leq o \leq O$, and $\cup_{o=1}^{O} R_o = \mathcal{R}$.

In this chapter, additionally, our objective is also to maintain the model cover as raw tuples are streamed into the system. This task involves adapting the model cover to the changes of the phenomenon that are observed over the region $\mathcal{R}$. To perform these tasks, we define a temporal dimension of the model cover. In our framework, a model cover is computed using the raw tuples in a time window of length H. Using H, we define a window of raw tuples as $\mathcal{B}_c = \langle b_i | cH \leq t_i \leq (c+1)H \rangle$, where c is a positive integer. Thus, $\mathcal{B}_c$ is a set of all the raw tuples b_i falling in the interval cH to $(c+1)H$. In addition, we write $g_i \sqsubset \mathcal{B}_c$ and $v_i \sqsubset \mathcal{B}_c$ to respectively denote the position $g_i = (x_i, y_i)$ and positioned value $v_i = (x_i, y_i, r_i)$ found in the raw tuple $b_i \in \mathcal{B}_c$.

Our focus is on estimating a model cover over the region $\mathcal{R}$ for the values in a time window $\mathcal{B}_c$. For clarity, let us concretely define the problem of model cover estimation:

PROBLEM 5.1: **Model Cover Estimation.** Given the region $\mathcal{R}$ and the window of raw tuples $\mathcal{B}_c$, compute the model cover $\mathcal{M}$, such that:

- It partitions/segments $\mathcal{R}$ into O regions $R_1, R_2, \ldots, R_O$ covering the region $\mathcal{R}$,
- It estimates models $M_1, M_2, \ldots, M_O$, such that each model corresponds to the region $R_1, R_2, \ldots, R_O$ respectively.

We propose various solutions for solving Problem 5.1. Broadly, the proposed solutions are of two types: (a) *non-adaptive* solutions that perform the partitioning and estimation using static policies, without iteratively improving the partitioning; and (b) *adaptive* solutions that perform the partitioning and estimation steps of Problem 5.1, using data characteristics and user-defined quality criteria (e.g., approximation error). In this chapter, we investigate two non-adaptive techniques, then, based on our observations, we propose two time- and space-efficient adaptive techniques that are able to accurately estimate the model cover $\mathcal{M}$ over a large geographical area.

Queries: To make our framework schematic complete, we show the query processing component in Figure 5.2. The queries consists of vehicles that register moving continuous queries. An example of such a query registered by a vehicle could be:

QUERY 5.1: **Moving Continuous Query**. Given the position $g = (x, y)$ of a vehicle, continuously return the concentration of NO_2 around it at an interval of 10 seconds.

These queries can be answered directly using the model cover $\mathcal{M}$ [24, 40, 110]. Note that although queries like Query 5.1 can be directly answered using the table `raw_tuples`, it is neither efficient nor accurate, since: (a) the number of raw tuples could be considerably large as compared to the number of models, and (b) the models minimize the errors caused during communication or due to the inherent imprecision of the sensors [24, 116]. Note that query processing is not the primary focus of this chapter; nonetheless, this component is shown in Figure 5.2 for presenting a complete picture of the ConDense framework.

Error Metric: The last foundational aspect is the error metric that we use in this chapter. Consider a model cover estimation method that partitions the window $\mathcal{B}_c$ into regions R_o where $1 \leq o \leq O$, such that $\mathcal{B}_c^o$ denotes the set of raw tuples b_i that are in region R_o. Suppose the model M_o approximates the value s_i with $\bar{s}_i$ then the error metric is defined as:

$$u_o = \frac{100}{|\mathcal{B}_c^o|} \sum_{v_i \sqsubset \mathcal{B}_c^o} u_o(v_i), \quad u_o(v_i) = \frac{|s_i - \bar{s}_i|}{\max(conc) - \min(conc)}, \tag{5.1}$$

where max(*conc*) and min(*conc*) are the upper and lower bounds of the normal concentration of the measured pollutant found in the urban environment. For example, if we are measuring Ozone, then the normal concentration of Ozone in urban environment is max(*conc*) = 0.15 ppm and min(*conc*) = 0.05 ppm (refer Table 5.1). We call the error metric in Eq. (5.1) the *normal percentage error*[3]. The normal percentage error compares the absolute approximation error with the normal value of a pollutant in the environment. Thus, the normal percentage error, intuitively, captures the impact of erroneous model approximations $\bar{s}_i$ on the quality of a model cover.

5.5 Non-Adaptive Methods for Model Cover Estimation

In this section we present the non-adaptive model cover estimation methods. Specifically, we investigate two strategies: first, a naive strategy in which the partitioning of $\mathcal{R}$ is performed by a rectangular division, second, we discuss a largely popular technique from the geo-statistics literature called Kriging. We observe that the non-adaptive methods are either computational expensive or inaccurate. In addition, as will be seen later, storing the model cover generated by these methods is also considerably expensive.

5.5.1 Grid-Based Model Cover

The Grid-based (GRIB) model cover estimation method is the most naïve strategy for estimating a model cover. This approach involves overlaying a grid over the region $\mathcal{R}$ and then estimating a linear regression model for individual grid elements. It simply divides the region $\mathcal{R}$ into a grid of a fixed size $\sqrt{O} \times \sqrt{O}$. Then each grid element forms the region R_o from Definition 5.1. Now the set of regions $R_1, R_2, \ldots, R_O$ induce a partition on the raw tuples in the window $\mathcal{B}_c$. Let us denote the set of raw tuples of the window $\mathcal{B}_c$ contained in the region R_o by $\mathcal{B}_c^o$. Now we can estimate a linear regression model M_o over the values $\mathcal{B}_c^o$ as:

$$s_i = \bar{s}_i + e_i, \quad \bar{s}_i = \alpha_0 + \alpha_1 x_i + \alpha_2 y_i. \tag{5.2}$$

Here, we estimate the parameters $(\alpha_0, \alpha_1, \alpha_2)$ by performing a least-squares fitting that minimizes the sum of e_i^2. The interpolation of the value at a position $g' = (x', y')$ is performed as:

$$\hat{s}(g') = \alpha_0 + \alpha_1 x' + \alpha_2 y'. \tag{5.3}$$

The main advantage of the GRIB model cover estimation method is that it is simple to implement. This simplicity comes from the static nature of the partitioning scheme; the partitioning scheme does not consider the characteristics of the underlying data. In the GRIB method, the granularity of the partitioning does not evolve temporally. Especially, for large geographical areas there could be a need to dynamically change

---

[3] We use *normal percentage error* and *approximation error* interchangeably.

the granularity and size of the partitioning based on the nature of the underlying phenomenon. For example, during peak hours of traffic, pollution is higher in downtown areas as compared to residential areas, and therefore we need a partitioning scheme that adapts to such change in behavior.

5.5.2 Kriging-Based Model Cover

The Kriging-based (KRIB) model cover estimation method is an approach that involves the use of Kriging [28]. Kriging is a well-known geo-statistical method for producing highly accurate models of data. In comparison to other interpolation approaches, Kriging has the advantage that it can also assign a confidence value to the interpolated values. These advantages (high accuracy and confidence values) naturally invite additional cost for creating and querying a Kriging-based model cover.

Kriging interpolates the value at position $g' = (x', y')$ by summing the weighted known values s_i as follows:

$$\hat{s}(g') = \sum_{i=1}^{|\mathcal{B}_c|} \lambda_i s_i, \quad \Upsilon(g_i, g') = \sum_{j=1}^{|\mathcal{B}_c|} \lambda_i \Upsilon(g_i, g_j), \tag{5.4}$$

where λ_i are the weights, such that $\sum_{i=1}^{|\mathcal{B}_c|} \lambda_i = 1$ and $\Upsilon(g_i, g_j)$ is the semi-variogram of the points g_i and g_j. λ_i are evaluated by solving the set of equations for $\Upsilon(g_i, g')$ where $1 \leq (i, j) \leq |\mathcal{B}_c|$. Additional details regarding Kriging can be found in [28].

Query processing time can be reduced by pre-computing the inverse matrix formed by $\Upsilon(g_i, g_j)$. Since $\Upsilon(g_i, g_j)$ is of size $|\mathcal{B}_c| \times |\mathcal{B}_c|$, storing the inverse of $\Upsilon(g_i, g_j)$ requires a large amount of memory. In Section 5.7, we find that even with pre-computation of the inverse of $\Upsilon(g_i, g_j)$, the KRIB model cover estimation method is not comparable with other model cover estimation approaches in answering point (interpolation) queries.

The Kriging method was introduced to efficiently approximate values when the sensors are stationary. But this method is not well suited for moving sensors, since in a mobile sensing environment the values along hotspots are excessively dense and should be condensed to reduce redundant sampling. Secondly, Kriging tries to fit a function to all the sensed values without eliminating redundant information, and, therefore has large overhead in terms of storage and computational complexity.

5.6 Adaptive Methods for Model Cover Estimation

In contrast to the non-adaptive techniques discussed in Section 5.5, the methods proposed in this section exploit the characteristics of the underlying data for obtaining a better partitioning of $\mathcal{R}$. In Section 5.7, we thoroughly compare the adaptive and non-adaptive methods, and experimentally establish the superiority of the adaptive techniques. Our adaptive techniques are based on unsupervised clustering algorithms. They intelligently partition $\mathcal{R}$ into regions, such that the models are always able to approximate the data with a certain error guarantee.

5.6.1 Adaptive DBSCAN

The *adaptive DBSCAN* method is a bottom-up clustering method, based on the well-known DBSCAN algorithm proposed in [45]. We first understand the reasons for the unsuitability of the DBSCAN algorithm for our problem; followed by the description of the adaptive DBSCAN method.

DBSCAN: Given a window of raw tuples $\mathcal{B}_c$, DBSCAN defines the density of $g_i \sqsubset \mathcal{B}_c$, denoted as $N_{Eps}(g_i)$, as the number of points that are present in a radius *Eps* around g_i. $g_i \sqsubset \mathcal{B}_c$ is called a *core point* if $N_{Eps}(g_i)$ is greater than $MinPts$, where $MinPts$ is a user-defined constant. All the points around g_i present in a radius *Eps* are called *directly density-reachable* from g_i.

A position g_j is *density-reachable* from g_i if there is a chain $(g_*)_1, \ldots, (g_*)_l$, where $(g_*)_1 = g_i$ and $(g_*)_l = g_j$, such that $(g_*)_2$ is directly density-reachable from $(g_*)_1$, $(g_*)_3$ from $(g_*)_2$, so on until $(g_*)_l$. Two positions g_i and g_j are *density-connected* if they are both density-reachable from a core point g_c. Now, we define $\mathcal{B}_c^o$ as a set of raw tuples, where $g_i \sqsubset \mathcal{B}_c^o$ is density-connected with $g_j \sqsubset \mathcal{B}_c^o$ for all $i \neq j$.

If a position g_i is not density-connected with any other points in $\mathcal{B}_c$, it is considered as noise and we set $\mathsf{c}(i) = NOISE$, where $\mathsf{c}(i) : i \mapsto o$ represents the cluster membership of a raw tuple b_i. By randomly selecting unclustered points (i.e., points where $\mathsf{c}(i) = UNCLASSIFIED$) and clustering all density-reachable tuples into the same region $\mathcal{B}_c^o$ we can divide the set $\mathcal{B}_c$ into O regions, where $0 \leq O \leq (|\mathcal{B}_c|/MinPts)$.

DBSCAN clusters the raw tuples only based on g_i and does not consider the sensor values s_i. Thus, it is possible that DBSCAN produces regions that cannot be modeled using polynomials having lower number of coefficients. To rectify this situation, we modify the DBSCAN algorithm, such that it produces regions that can be modeled using lower number of coefficients. We call this modified algorithm *Adaptive DBSCAN*.

Adaptive DBSCAN: In the Adaptive DBSCAN (Ad-DBS) method we continuously maintain a linear regression model M_o (refer Eq. (5.2)) for each region R_o. In addition, we provide the following modified definition for density-reachable and density-connected:

DEFINITION 5.2: **Model Density-Reachable.** A positioned value v_i is model density-reachable from $v_j \sqsubset \mathcal{B}_c^o$, if position g_i is density-reachable from g_j and $u_o(v_j) < \tau_r$, where τ_r is a user-defined quality threshold, u_o is the error metric and $cH \leq t_i, t_j \leq (c+1)H$.

DEFINITION 5.3: **Model Density-Connected.** Positioned value v_i and v_j are model density-connected if v_i and v_j are model density-reachable from $v_l \sqsubset \mathcal{B}_c^o$.

Algorithm 5.1 performs the partitioning of $\mathcal{B}_c$, such that each positioned value $v_i \sqsubset \mathcal{B}_c^o$ is model density-connected to $v_j \sqsubset \mathcal{B}_c^o$ for all $i \neq j$. The function CHECKERRORANDADD temporarily adds v_j to $\mathcal{B}_c^o$ and re-computes the model M_o. If $u_o(v_j) > \tau_r$, then v_j is not model density-connected to the other tuples in $\mathcal{B}_c^o$, therefore it is not permanently added

Algorithm 5.1 The adaptive DBSCAN algorithm.

Input: Window $\mathcal{B}_c$, error threshold τ_r, Eps, $MinPts$.

Output: Number of regions O, regions R_o and a linear regression model M_o for each region respectively where $o = 1, \ldots, O$.

```
 1: $o \leftarrow 1$
 2: for all $v_i \sqsubset \mathcal{B}_c$ do
 3:     if $c_i = UNCLASSIFIED$ then
 4:         if expandCluster($v_i$,$o$) then
 5:             $o \leftarrow o + 1$
 6: procedure ExpandCluster($v_i$,$o$) : boolean
 7:     $seeds \leftarrow$ regionSearch($v_i, Eps$) $\setminus v_i$
 8:     if $|seeds| < MinPts$ then
 9:         $c_i \leftarrow NOISE$
10:         return $false$
11:     else
12:         Add($M_o$,$v_i$)
13:         for all $v_j \in seeds$ do
14:             if checkErrorAndAdd($M_o$,$v_j$) $\neq$ success then
15:                 $seeds \leftarrow seeds \setminus v_j$
16:         while $|seeds| \neq 0$ do
17:             $v_j \leftarrow removeOneValue(seeds)$
18:             $results \leftarrow$ regionSearch($v_j, Eps$) $\setminus v_j$
19:             if $|results| > MinPts$ then
20:                 for all $v_f \in results$ do
21:                     if $c_f = UNCLASSIFIED$ then
22:                         $seeds \leftarrow seeds \cup c_f$
23:                     if $c_f \in \{NOISE, UNCLASSIFIED\}$ then checkErrorAndAdd($M_o$,$v_f$)
24:         return true
```

to $\mathcal{B}_c^o$. In Step 7, regionSearch returns the points in a radius Eps around v_i, and in Step 12, Add unconditionally adds v_i to M_o.

Interpolation using Ad-DBS: Because of the new definitions of model density-reachable and density-connected it may happen that the regions R_o produced by the Ad-DBS method overlap with each other. Therefore, for interpolating the value $\hat{s}(g')$ at position (x', y') it is unclear whether one or more regions R_o should be used. To solve this problem, we introduce a weighting scheme (refer Figure 5.3) that produces the interpolated value $\hat{s}(g')$ by assigning weighting functions $K_o(g')$ to the regions R_o, such that:

$$\hat{s}(g') = \sum_{o=1}^{O} \kappa_o(g')\hat{s}_o(g'), \tag{5.5}$$

where $\kappa_o(g') = \frac{K_o(g')}{\sum_{h=1}^{O} K_h(g')}$ and $\hat{s}_o(g')$ is the interpolated value using model M_o.

Since the normal percentage error metric introduced in Eq. (5.1) does not consider overlapping regions, we introduce the following modified version of the normal percentage error for analyzing this weighting scheme of the Ad-DBS method:

$$\hat{u}_o = \frac{100}{|\mathcal{B}_c^o|} \sum_{v_i \sqsubset \mathcal{B}_c^o} \hat{u}_o(v_i), \quad \hat{u}_o(v_i) = \frac{|s_i - \hat{s}_i(g_i)|}{\max(conc) - \min(conc)}. \tag{5.6}$$

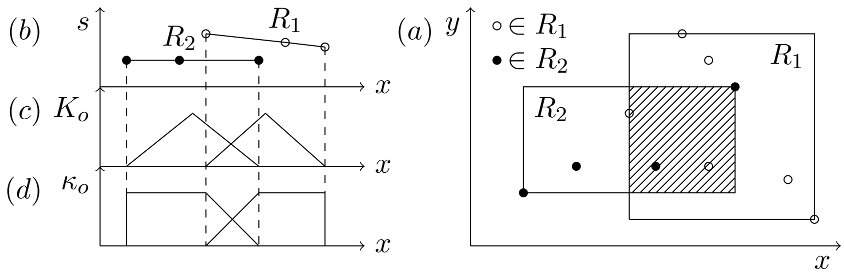


Figure 5.3: Weighting scheme for Ad-DBS. (a) shaded area shows an example of two overlapping regions, (b) shows the regions with the corresponding sensor values s, (c) and (d) present the weighting functions K_o and κ_o used for interpolation.

Notably, the difference between u_o and $\hat{u}_o$ characterizes the error introduced by the weighting scheme used in the Ad-DBS method.

5.6.2 Adaptive K-Means

In this section we start by discussing the k-means clustering method (a top-down clustering approach), then briefly discuss the reasons why the vanilla k-means clustering method cannot be used for obtaining a model cover with a user-defined approximation error threshold. Then we propose the adaptive k-means model cover estimation method that overcomes the shortcomings of the the k-means clustering method and efficiently produces an highly accurate model cover.

K-means Clustering: Given the raw tuples in a window $\mathcal{B}_c$ and the number of clusters O, the objective of the k-means clustering method is to divide the raw tuples in the window $\mathcal{B}_c$ into k sets $\mathcal{B}_c^1, \mathcal{B}_c^2, \ldots, \mathcal{B}_c^O$ such that the following objective function is minimized:

$$\arg\min_{\hat{\mu}_o} \sum_{o=1}^{O} \sum_{v_j \sqsubset \mathcal{B}_c^o} ||v_j - \hat{\mu}_o||, \tag{5.7}$$

where $\hat{\mu}_o = (x_o, y_o, r_o)$ is known as the centroid of the partition $\mathcal{B}_c^o$. Then the region R_o is the region that surrounds points in $\mathcal{B}_c^o$, and the model cover can be obtained by computing a regression model M_o for each $\mathcal{B}_c^o$.

The k-means clustering method does not achieve our objective of partitioning the raw values $\mathcal{B}_c$, since the euclidean distance used by the k-means method may compensate a large difference in the sensor value s with a small difference in the position (x, y). On the contrary, our objective is that values in a particular region R_o should be close in the position *and* in the sensor value. Moreover, another requirement is that the raw tuples $\mathcal{B}_c$ should be approximated within a user-defined normal percentage error threshold τ_n. For achieving these objectives we propose an adaptive variant of the k-means clustering method.

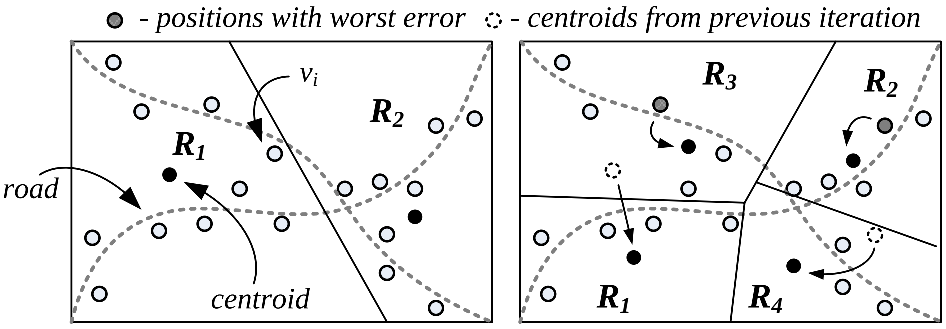
Adaptive K-means: The algorithm used by adaptive k-means (Ad-KMN) method is shown in Algorithm 5.2. Figure 5.4 shows an example of the Ad-KMN method on toy data.

Algorithm 5.2 The adaptive k-means model cover method.

```
Input: Window $\mathcal{B}_c$, error threshold $\tau_n$.
Output: Number of regions $O$, regions $R_o$ and a linear regression model $M_o$ for each region
    respectively where $o = 1, \ldots, O$.
$newCluster \leftarrow true$
$clusterChanged \leftarrow true$
$\mu_1 \leftarrow \texttt{rand}(\mathcal{B}_c)$        ▷ Choose a random position as the initial cluster center
while $newCluster$ do
    $newCluster \leftarrow false$
    while $clusterChanged$ do
        $clusterChanged \leftarrow false$
        for $o$ in 1 to $O$ do
            $\mu_o^* \leftarrow recenter(\mathcal{B}_c^o)$        ▷ Re-compute the cluster center of region $R_o$
            if $\mathcal{B}_c^o \neq \mathcal{B}_c^{o*}$ then        ▷ Re-compute $\mathcal{B}_c^o$ by considering $\mu_o^*$. Lets denote it as $\mathcal{B}_c^{o*}$
                $clusterChanged \leftarrow true$
            $\mu_o \leftarrow \mu_o^*$
    $O^* \leftarrow O$
    for $o$ in 1 to $O$ do
        $M_o, u_o, \mu_o^\bullet \leftarrow estimateModel(\mathcal{B}_c^o)$        ▷ Find the point $\mu_o^\bullet$ producing worst error
        if $u_o > \tau_n$ then
            $O^* \leftarrow O^* + 1$, $\mu_{O^*} \leftarrow \mu_o^\bullet$
            $newCluster \leftarrow true$
    $O \leftarrow O^*$
```

Assume that before executing the Ad-KMN method, we compute two k-means centers μ_1 and μ_2 over all the positions $g_i \sqsubset \mathcal{B}_c$. A snapshot after this step is shown in Fig. 5.4(a). Next, we check whether the errors u_1 and u_2 are within a user-defined threshold τ_n. The principle here is to introduce an additional cluster centroid $\mu_o^\bullet$ for each region R_i where $u_o > \tau_n$, by choosing the g_i that produced the worst error for R_i. Suppose, both R_1 and R_2 of Fig. 5.4(a) violate the error condition (i.e., $u_1 > \tau_n$ and $u_2 > \tau_n$), then we initialize two new centroids $\mu_1^\bullet$ and $\mu_2^\bullet$ and we re-adjust the four centroids (μ_1, μ_2, $\mu_3 = \mu_1^\bullet$ and $\mu_4 = \mu_2^\bullet$), by executing the standard k-means algorithm on the four centroids. The result of this step is shown in Fig. 5.4(b).

As will be shown in Section 5.7, the Ad-KMN method exhibits fast convergence


● - positions with worst error
◌ - centroids from previous iteration
v_i
R_1
R_2
road
centroid
R_3
R_2
R_1
R_4

(a) (b)

Figure 5.4: Ad-KMN iterations on toy data. (a) the centroids of regions R_1 and R_2 are computed, after which models M_1 and M_2 are estimated. (b) since error $u_1 > \tau_n$ and $u_2 > \tau_n$, we add two new clusters R_3 and R_4 using k-means clustering algorithm.

characteristics. In addition, the Ad-KMN method also requires lower storage space and can produce accurate model covers.

5.6.3 Efficiently Maintaining the Model Cover

Furthermore, we are interested in maintaining the model cover as new windows $\mathcal{B}_c$ are streamed into ConDense. Specifically, given several windows of raw values $\mathcal{B}_c$ where $c = (1, 2, \ldots, C)$, we are interested in continuously maintaining the model cover while minimizing the number of additional computations required for model cover maintenance.

We start by estimating the cluster centroids μ_o over a training window $\mathcal{B}_D$ of size $D \gg H$ using the adaptive method. The adaptive method returns the regions R_o and models M_o where $o = (1, \ldots, O)$. Now, assume that the first window of new raw values $\mathcal{B}_c$ is available. $\mathcal{B}_c$ is first partitioned according to the cluster centers μ_o, such that $\mathcal{B}_1^o$ contains the raw tuples where $||g_i - \mu_o||$ is minimal.

Next, we compute the error metric u_o for $\mathcal{B}_c^o$. If u_o is greater than a user-defined threshold τ_r, then we invalidate the model M_o and re-estimate its coefficients. We perform a similar test for all the other $\mathcal{B}_c^o$. We use flops[4] to measure the cost of updating the model M_o. Suppose the cost of updating the window $\mathcal{B}_c$ be denoted as $\mathcal{C}(\mathcal{B}_c)$, then it can be computed as follows [52]:

$$\mathcal{C}(\mathcal{B}_c) = \sum_{\forall o \ s.t. \ u_o > \tau_r} 2 \cdot |M_o|^2 \left(|\mathcal{B}_c^o| - \frac{|M_o|}{3} \right), \tag{5.8}$$

where $|M_o|$ is the number of coefficients to estimate for the model M_o. In our case, $|M_o| = 3$ since M_o has three coefficients $(\alpha_0, \alpha_1, \alpha_2)$. The better the adaptive method partitions the region $\mathcal{R}$, the less would be the cost of maintaining the model cover, since the adaptive method would have found areas having similar data distributions. Therefore, the raw tuples that are newly streamed into the system in a reasonably short interval do not require a model update, resulting in potentially dramatic saving of computation required for model cover maintenance.

As we will show in Section 5.7, such a strategy of adaptively maintaining the model cover is effective and can yield up to *approximately 3x less number of flops* (for Ad-KMN) as compared to using the same strategy over a GRIB model cover, thereby establishing the advantages of using an adaptive method for model cover estimation.

5.7 Experimental Evaluation

In this section we perform extensive experimental evaluation of the various model cover estimation approaches. In Section 5.7.1 we compare the model cover estimation approaches with respect to the normal percentage error. In Section 5.7.2, we compare the efficiency of the adaptive and non-adaptive techniques for model cover estimation in

---

[4] A *flop* represents either the addition or the multiplication of two floating point numbers.

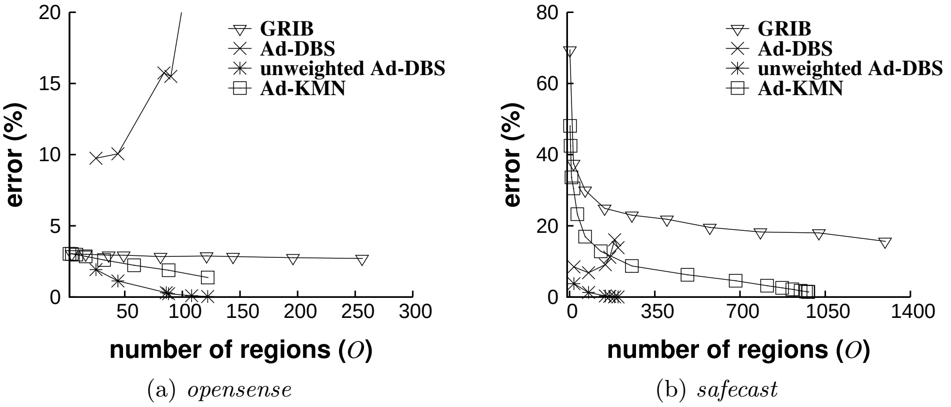
terms of the storage space and estimation time. Lastly, Section 5.7.3 compares adaptive and non-adaptive methods with respect to their temporal model cover validity characteristics. For all the experiments we use the *opensense* and the *safecast* datasets described in Section 5.2.

5.7.1 Error Analysis

We start by analyzing the different model cover estimation approaches using the normal percentage error defined in Eq. (5.1). Figure 5.5 shows the error as the number of regions are increased for the GRIB, Ad-KMN, and Ad-DBS methods. The process of adding more regions terminates when the error is less than the user-defined error threshold $\tau_n = 1\%$ or adding new regions does not significantly reduce the error. For this experiment the size of the window $\mathcal{B}_c$ is set to 6 hours. Clearly, for all the three approaches the percentage normal error decreases with increase in the number of regions.

Specifically, for *safecast* the Ad-KMN method delivers an *improvement of 12.5 times less error* as compared to the GRIB method for $O = 1000$. In contrast, for *opensense*, the Ad-KMN method does not show significant improvements (2.1 times less error for $O = 120$) over the GRIB method. This is because, as described earlier, *opensense* data does not exhibit high spatial-temporal variation. Therefore all the methods are able to achieve lower error. In general, the adaptive methods have lower number of regions as compared to the non-adaptive methods. For example, for *safecast*, the GRIB method has 1296 regions as compared to the 981 regions of the Ad-KMN method at convergence (refer Figure 5.5).

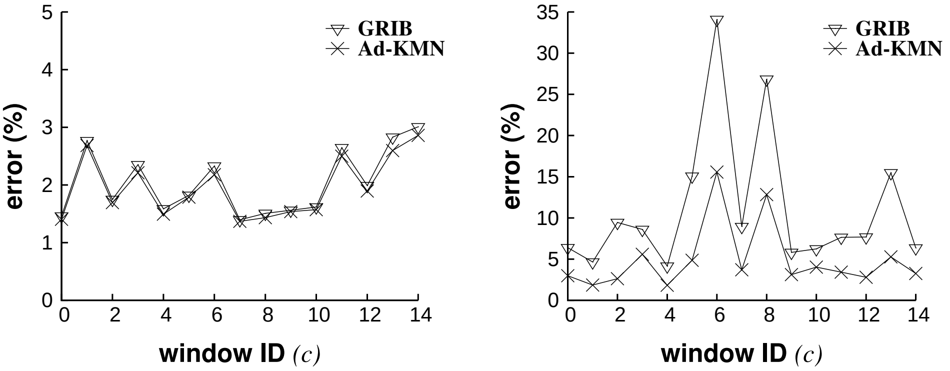
Additionally, to substantiate the results in Figure 5.5, we plot the error for 15 randomly chosen windows $\mathcal{B}_c$ for the Ad-KMN and GRIB methods where the maximum number of regions is $O = 50$ and is constant. Similar observations to Figure 5.5 could be made in Figure 5.6. For *safecast* the improvement obtained by using the Ad-KMN



(a) *opensense* (b) *safecast*

Figure 5.5: Comparing the decrease in percentage error as the number of regions increase. *Unweighted Ad-DBS* denotes Ad-DBS without the weighting scheme of Eq. (5.5). Note the different ranges on the y-axis.

method as compared to the GRIB method is significantly higher than *opensense*. In Figure 5.6, we do not show the result for the Ad-DBS method, since for the Ad-DBS method, it is impossible to control the number of regions that will be created, thus leading to an unfair comparison. These experiments clearly establish that adaptive methods, like Ad-KMN, can dramatically reduce the error as compared to the non-adaptive methods.



(a) *opensense* (b) *safecast*

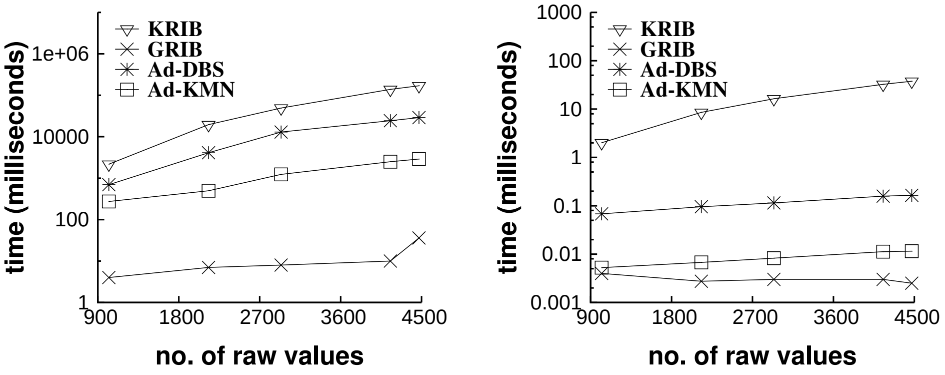
Figure 5.6: Comparing the percentage normal error for Ad-KMN and GRIB over randomly chosen windows $\mathcal{B}_c$. Note the different ranges on the y-axis.

Note that Figure 5.5 does not show the KRIB method, since the KRIB method always produces zero error due to the fact that Kriging always finds a function that passes perfectly through the given points. The zero error of Kriging comes at a cost: estimating and storing a Kriging model is substantially inefficient (refer Section 5.7.2) as compared to the adaptive methods, and therefore is not suitable for a database environment.

In Figure 5.5 the Ad-DBS method produces higher error as compared to the Ad-KMN method. The reason for such behavior is that, the increase in error due to the over-simplified weighting scheme of the Ad-DBS method (see Eq. (5.5)), is more as compared to the decrease in error obtained by adding more regions; thus leading to an overall error increase. To experimentally establish this observation, in Figure 5.5 we also show the normal percentage error obtained by the Ad-DBS method without the weighting scheme of Eq. (5.5). This shows that an appropriate choice of the weighting scheme is important for the Ad-DBS method.

5.7.2 Comparing Efficiency of Model Cover Estimation Methods

Next, we compare the time- and space-efficiency of the model cover estimation methods. Fig. 5.7(a) compares the average time required for model cover estimation using different methods. Fig. 5.7(b) compares the average time required for processing a point query. Here a *point query* is defined as a query that requests for the interpolated value at a particular position $g = (x, y)$. The average point query processing time is computed over 4000 point queries in the region $\mathcal{R}$.

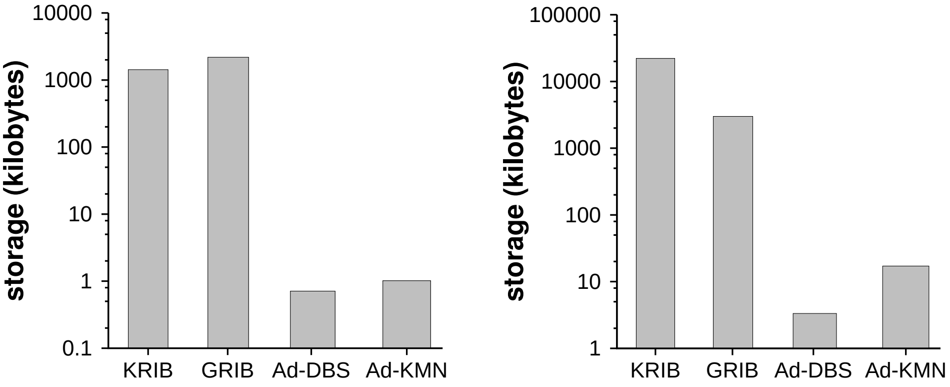


(a) model cover estimation (b) query processing

Figure 5.7: Comparing efficiency of (a) model cover estimation and (b) processing a point query (interpolation) on *opensense*.

On the one hand, the most time-efficient method for model cover estimation is the GRIB method, on the other hand it is significantly inefficient in terms of space (refer Figure 5.8). Moreover, the Ad-KMN method requires 1160 times less memory as compared to the GRIB method, and can be estimated by spending on an average 1.5 seconds or 80 times more time than the GRIB method.

Obviously, the KRIB method is significantly time- and space-inefficient as compared to the other model cover estimation methods, demonstrating that the KRIB method is clearly not usable in a database environment. Lastly, the Ad-DBS method can be stored using slightly less memory, but exhibits less efficiency in processing a point query as compared to the Ad-KMN method, and as seen in Section 5.7.1 it produces high normal percentage error.



(a) *opensense* (b) *safecast*

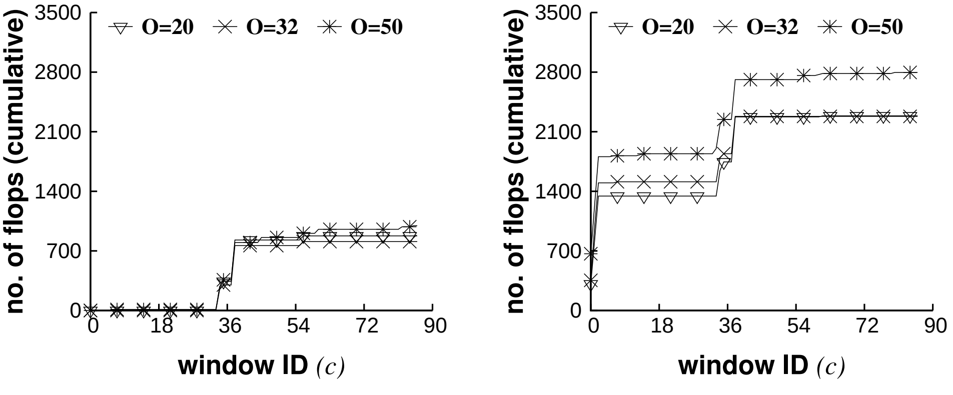
Figure 5.8: Comparing the memory requirement of all the model cover estimation methods.

5.7.3 Analyzing Temporal Validity of Model Cover

We perform the last set of experiments on *opensense*. These experiments are performed to compare the temporal validity characteristics between adaptive and non-adaptive model cover estimation methods. Particularly, we zone into comparing temporal behavior of the GRIB and the Ad-KMN methods.

We start by choosing a region $\mathcal{R}' \subset \mathcal{R}$. From the raw tuples in $\mathcal{R}'$, we choose a training window $\mathcal{B}_D$ of size 6 hours and 88 testing windows $\mathcal{B}_c$ of size 30 minutes. Note that $\mathcal{B}_D$ and $\mathcal{B}_c$ are consecutive in time. Then we choose the model retain threshold (τ_r) as 1% and apply the algorithm for maintaining the model cover from Section 5.6.3 and compute the cost $\mathcal{C}(\mathcal{B}_c)$ for each window $\mathcal{B}_c$. To substantiate our experiment, we choose three different values of O for the Ad-KMN method and adjust the GRIB method so that the number of used grid cells by the GRIB method are always equal to that of the Ad-KMN method.

Figure 5.9 shows the cumulative number of flops required to maintain the model cover. Admittedly, the Ad-KMN method requires a factor 2.7 less number of flops as compared to the GRIB method. In conclusion, the regions R_o that are produced by the Ad-KMN method are valid for a longer time, thus require less number of flops. For example, the Ad-KMN method requires zero flops for the first 34 windows as opposed to the 1874 flops required by the GRIB method.



(a) Ad-KMN (b) GRIB

Figure 5.9: Comparing temporal validity of the model cover produced by (a) Ad-KMN and (b) GRIB on *opensense*.

5.8 Conclusion

In this chapter, we presented non-adaptive and adaptive techniques for managing data produced by a CGSN. Our experiments showed that the adaptive model cover estimation methods perform significantly better that the non-adaptive methods. The non-adaptive methods are either inaccurate or are memory inefficient. Overall, the adaptive k-means method exhibits acceptable tradeoffs between the proposed approaches. Computing the model cover using the adaptive k-means (Ad-KMN) algorithm takes slightly more time as compared to the GRIB method, but it dramatically reduces the number of models and, therefore, the memory required to store these models.

Chapter 6

Conclusion and Future Directions

In everything... uniformity is undesirable. Leaving something unfinished makes it interesting, and gives one the feeling that there is room for growth... Even when building the imperial palace, they always leave one place unfinished.

Japanese Essays in Idleness,
14th Century

6.1 Conclusion

In this thesis we proposed several methods for querying and managing time-series data. We proposed in-depth solutions for several problems regarding time-series data. Particularly, we focused on computing statistical measures, generating useful attributes for uncertain data, and managing and querying participatory sensing data.

For the first time, we defined and proposed the notion of affine relationships for computing and querying several statistical measures using an unified approach. We proposed the affine clustering algorithm for clustering the time-series data, such that high-quality affine relationships could be found. We proposed the SYMEX and SYMEX+ algorithms that are capable of computing affine relationships in linear time. We demonstrated that the SCAPE index structure can easily index all the statistical measures and produce orders of magnitude improvement in efficiency for processing measure threshold and range queries, as compared to the naive methods and methods proposed in the literature for this problem.

We recognized that there is a lack of methods for generating probabilistic databases. Moreover, a large variety of applications that are built on (imprecise) time series are still incapable of enjoying the benefits from well-established tools for processing probabilistic databases. As a solution to this problem, we proposed a novel and generic solution for creating probabilistic databases from imprecise time-series data. Our proposal included two novel components: the dynamic density metrics that effectively infer time-dependent probability distributions for time series and the Ω–View builder that uses the inferred distributions for creating probabilistic databases. We also introduced the σ–cache that enables efficient creation of probabilistic databases while obeying user-defined constraints. Comprehensive experiments highlight the effectiveness of our approach.

Lastly, we proposed methods for managing and querying community-sensed data. We proposed non-adaptive and adaptive techniques for managing data produced by a CGSN. Our experiments established that the adaptive model cover estimation methods, which use dynamic partitioning approaches, demonstrate promising performance gains as compared to the non-adaptive methods. Particularly promising was the adaptive k-means (Ad-KMN) model cover estimation method, since it showed the best model cover quality, considering other parameters, like storage, computational cost, and temporal validity.

6.2 Future Directions

We recognize that the work described in this thesis can be strengthened in a number of ways and we suggest the following as future work.

6.2.1 AFFINITY

We are planning to extend the AFFINITY framework (Chapter 3) in the following directions:

- **Pruning affine relationships:** It is not mandatory that we process and store all the affine relationships. We can, if required, prune the unnecessary affine relationships on the basis of domain knowledge, query requirements, low correlation between a sequence pair, etc. Such pruning techniques will be considered in subsequent works. On the contrary, here we consider all the affine relationships returned by the SYMEX algorithm, for clearly demonstrating performance and scalability results.

- **Dynamic affine relationships:** Affine relationships can change dynamically, especially as new data is streamed into the system. Handling dynamic affine relationships requires: (i) a sequentially updating version of the AFCLST algorithm [15], and (ii) updating the changed affine relationships in the SCAPE index. Task (ii) is similar to a standard index update operation in a DBMS. Supporting dynamic affine relationships is an interesting direction that we plan to explore in our subsequent works.

- **Distributed query processing:** Many datasets are large and cannot be stored on a single computing device. Therefore, researching techniques for distributing the SCAPE index, and for performing affine clustering in a distributed setting becomes important. Thus, extending the proposed techniques to a distributed environment is an open problem.

6.2.2 Creating Probabilistic Databases

In Chapter 4 we proposed a framework for efficiently creating probabilistic databases. There are many issues that remain to be researched, below we briefly discuss some future directions:

- **Creating multivariate probabilistic databases:** In Chapter 4, we assumed that the data obtained from the time-series data sources is univariate. Thus, in the current framework, multivariate time-series data would be treated as a collection of univariate time-series data. This assumption ignores the correlation that exists in many real-world multivariate time-series datasets.

  The problem of representing and modeling correlation is important and has numerous practical applications. But, this problem is extremely challenging especially for high-dimensional data. In many cases, however, the data that is encountered is not high-dimensional, for example, GPS trajectories are only 2-dimensional. Taking this into consideration, we have proposed solutions for creating probabilistic databases over low-dimensional data in [65]. Our future works will focus on the general problem, along with efficient caching mechanisms (like, the σ-cache) for multi-dimensional data.

- **Processing probabilistic queries:** A standard method proposed in existing literature for processing probabilistic queries does not assume a particular form of distribution for the data generated by the imprecise time-series data sources. However, as briefly discussed in Appendix 4.A, in certain cases making such a distributional assumption may dramatically increase the efficiency of processing probabilistic queries. We plan to further investigate this direction for proposing efficient query processing methods.

- **Caching multivariate Gaussian distributions:** In Section 4.6.1 we introduced the σ–cache for caching Gaussian distributions. We also derived its parameters given user-defined accuracy and memory constraints. The current version of the σ–cache is designed for univariate Gaussian distributions. We plan to extend these ideas to a multivariate version of the cache, at the same time, deriving useful and provable guarantees.

6.2.3 ConDense

In relation to the ConDense framework (Chapter 5), we propose the following as future work:

- **Complete re-learn of model cover:** In our approach for handling temporal evolution of the model cover (refer Section 5.6.3), we have not considered a complete re-learn of the model cover if the cost $\mathcal{C}(\mathcal{B}_c)$ increases dramatically. On the one hand, re-learning could reduce the cost $\mathcal{C}(\mathcal{B}_c)$ for future windows $\mathcal{B}_c$, but on the other hand, could incur down-time for the system. Another alternative to complete re-learn is to develop techniques that merge/split the models, such that a reasonable model cover is always maintained. We plan to explore the trade-off between complete re-learn and merge/split in our future works.

- **Continuous query processing:** The ConDense framework describes the continuous query processing component, and evaluates query costs with respect to model cover techniques. As a next natural step, we plan to investigate efficient and accurate query processing solutions. This, we believe, will open-up interesting research issues like, query optimization, response caching, model cover indexing, etc.

- **Utility-driven sampling:** If we relax the autonomous sensing assumption in the community sensing paradigm, then there is an issue of utility-driven sampling. Here, the underlying phenomenon is sampled only as much as required by a given set of continuous queries. The utility is defined by the queries based on the accuracy guarantee requirements provided by the user.

- **Online adaptive k-means:** Finally, the current adaptive k-means algorithm, as described in Section 5.6.2, operates on a batch of tuples $\mathcal{B}_c$ and produces a model cover. The model cover estimate can only be updated after processing the entire window $\mathcal{B}_c$. Changing the adaptive k-means algorithm, such that it can update the model cover even after a single tuple is streamed into the system is an interesting direction. We plan to explore this direction in our subsequent works.

List of Symbols

$\bar{s}_i$	Approximation of the time-series data value s_i, page 107
$\mathrm{P}_l(R_l)$	Cumulative probability distribution function of R_l at time l, page 78
$\mathrm{P}_l(R_l;\hat{\theta}_l)$	Cumulative probability distribution of R_l with parameters $\hat{\theta}_l$, page 87
δ	User-defined threshold $\delta \in \mathbb{R}$, page 18
$\mathsf{D}(\mathbf{S})$	Derived measure of the matrix $\mathbf{S}$, page 48
ϵ	Approximation error bound, page 32
$\hat{s}_l, \mathrm{E}(R_l)$	Expected true value at time l, page 78
$\mathsf{C}(\mathbf{S})$	Dispersion measure of the matrix $\mathbf{S}$, page 48
κ	Scaling factor of the ARMA-GARCH and Kalman-GARCH dynamic density metrics, page 82
Λ_l	A set of probabilities $\{o_1, \ldots, o_U\}$, page 78
$\lceil x \rceil$	A smallest integer value that is not smaller than x, page 78
$\mathsf{L}(\mathbf{S})$	Location measure of the matrix $\mathbf{S}$, page 48
$\mathbf{O}_p$	Pivot pair matrix corresponding to pivot pair p, page 55
$\mathbf{S}$	Data matrix $\mathbf{S} = [s_1, s_2, \ldots, s_n] \in \mathbb{R}^{m \times n}$, page 6
$\mathbf{S}_e$	Sequence pair matrix corresponding to sequence pair e, page 48
$\mathcal{N}(\hat{s}_l, \hat{\sigma}_l^2)$	Gaussian density function at time l with mean $\hat{s}_1$ and variance $\hat{\sigma}_1^2$, page 81
$\mathcal{N}(\mu, \sigma^2)$	Gaussian density function with mean μ and variance σ^2, page 78
$\max(s_j)$	Upper bound of an uncertainty range, page 31
$\max(s_l)$	Upper bound computed by the ARMA-GARCH and Kalman-GARCH algorithms, page 82

$\mathcal{C}(\mathcal{B}_c)$	Cost of updating the model M_o for window $\mathcal{B}_c$, page 114
$\mathsf{B}_j(\mathcal{W})$	Confidence in the time-series data source w_j, page 17
$\mathsf{c}(v)$	Cluster assignment function that returns the cluster identifier for the entity v, page 54
$\mathsf{F}(\mathbf{X}, \mathbf{Y})$	Least significant Frobenius distance between matrices $\mathbf{X}$ and $\mathbf{Y}$, page 52
$\min(s_j)$	Lower bound of an uncertainty range, page 31
$\min(s_l)$	Lower bound computed by the ARMA-GARCH and Kalman-GARCH algorithms, page 82
$\mu_o, \mu_o^\bullet$	Current and newly added cluster centroids of the region R_o, page 113
$\nu^2(\boldsymbol{v})$	Sample variance of the vector $\boldsymbol{v}$, page 84
ν_l^2	Sample variance in the sliding window $\boldsymbol{s}_{l-1}^H$, page 80
Ω	A set of ranges $\{\omega_1, \ldots, \omega_U\}$ for creating probabilistic database, page 78
$\mathbb{P}_l(R_l)$	Probability density function of R_l at time l, page 78
ψ	Queried series identifiers of a measure computation query, page 49
$\rho(\mathbf{S})$	Correlation coefficient matrix of the matrix $\mathbf{S}$, page 48
$\mathcal{A}_R$	Response of a measure range query, page 49
$\mathcal{A}_T$	Response of a measure threshold query, page 49
$\mathcal{B}_c$	c^{th} window of size H consisting of raw tuples $\langle b_i \mid cH \leq t_i \leq (c+1)H \rangle$, page 107
$\mathcal{B}_c^o$	The set of raw tuples b_i that are in region R_o, page 107
$\mathcal{E}$	Time stamps in feasible regions, page 29
$\mathcal{F}_{i-1}$	Information available until time $i-1$., page 81
$\mathcal{G}^k$	k^{th} segment in a data stream containing tuples $((t_{i_{k-1}+1}, s_{i_{k-1}+1}), \ldots, (t_{i_k}, s_{i_k}))$, page 32
$\mathcal{I}, \mathcal{P}$	Series identifier and sequence pair sets, page 48
$\mathcal{M}$	Model cover for region $\mathcal{R}$ consisting of models $M_1, \ldots, M_O$, page 106
$\mathcal{R}$	Geographical region composed of sub-regions $R_1, \ldots, R_O$, page 105
$\mathcal{W}$	Time-series database consisting of sources w_j, where $j = (1, \ldots, n)$, page 6
$\Upsilon(g_i, g_j)$	The semi-variogram of the positions g_i and g_j, page 109

φ_e	Normalizer of for sequence pair e, page 50
$\mathit{\Pi}(\mathbf{S})$	Dot product matrix of the matrix $\mathbf{S}$, page 49
ς	Maximum consecutive erroneous values, page 84
$\mathit{\Sigma}(\mathbf{S})$	Covariance matrix of the matrix $\mathbf{S}$, page 49
$\boldsymbol{r}_i$	Row vector of all the data values observed at time t_i, such that $r_i \in \mathbb{R}^n$, page 6
$\boldsymbol{s}_{t-1}^H$	Sliding window having H values in the range $[l-H, l-1]$, page 78
$\boldsymbol{s}_j$	Column vector of all the data values observed by the data source w_j, such that $s_j \in \mathbb{R}^m$, page 6
ξ_{qd}	Scale projection of $\boldsymbol{\beta}_{qd}$ on $\boldsymbol{\alpha}_q$, page 61
b_i	Raw tuple (t_i, x_i, y_i, s_i) from a moving sensor, page 106
D	Degree of polynomial regression, page 22
e_i	Prediction errors or least-squares fitting error, page 18
f_{dec}	Decoding function, page 20
f_{enc}	Encoding function, page 20
$g_i \sqsubset \mathcal{B}_c$	Position $g_i = (x_i, y_i)$ found in the raw tuple b_i, page 107
g_i	Position (x_i, y_i) of a moving sensor at time t_i, page 106
g_{ij}	Transmission message generated after encoding, page 20
L	Seasonal period of the SARIMA model, page 18
o_u	The probability that $\omega_u \in \Omega$, page 78
o_{i1}^1, s_{i1}^1	Weight o_{i1}^1 of the value v_{i1}^1 used in particle filtering, page 29
p	Number of particles in a particle filter, page 29
R_{ij}	Random variable associated with the data value s_{ij}, page 6
R_i	Random variable associated with the row vector r_i, page 6
s_{ij}	Data value observed by the source w_j at time t_i, such that $s_{ij} \in \mathbb{R}$, page 6
u_o	Normal percentage error computed for region R_o, page 108
$v_i \sqsubset \mathcal{B}_c$	Positioned value $v_i = (x_i, y_i, r_i)$ found in the raw tuple b_i, page 107
v_i	Positioned value (x_i, y_i, s_i) from a moving sensor, page 106

w_j	Time-series data source in the database $\mathcal{W}$, page 6
y_i	Probability integral transform of s_i *w.r.t.* $\mathbb{P}_i(R_i)$, page 78
z_i	White noise drawn from a zero mean multi-variate Gaussian distribution, page 25
ARMA(α,β)	ARMA model of order (α, β), page 80
GARCH(ζ,η)	GARCH model of order (ζ, η), page 82

Bibliography

[1] National Ambient Air Quality Standards (NAAQS). `http://www.epa.gov/air/criteria.html`. 103

[2] The Safecast project. `http://blog.safecast.org/`. ii, 104

[3] Urban Atmosphere Project. `http://www.urban-atmospheres.net/`, 2006. 105

[4] AERMOD (EPA). http://www.epa.gov/scram001/dispersion_prefrec.htm, 2009. 104, 105

[5] K. Aberer, S. Sathe, D. Chakraborty, A. Martinoli, G. Barrenetxea, B. Faltings, and L. Theile. OpenSense: Open community driven sensing of environment. In *IWGS (along with ACM GIS)*, 2010. ii, 101, 103

[6] R. Agrawal, C. Faloutsos, and A. Swami. Efficient similarity search in sequence databases. In *Foundations of Data Organization and Algorithms*, pages 69–84, 1993. 2, 13, 39, 41, 71

[7] R. Agrawal, K. Lin, H. Sawhney, and K. Shim. Fast similarity search in the presence of noise, scaling and translation in time-series databases. In *VLDB*, 1995. 71

[8] Y. Ahmad and S. Nath. COLR-Tree: Communication-efficient spatio-temporal indexing for a sensor data web portal. In *ICDE*, pages 784–793, 2008. 2

[9] G. Anastasi, M. Conti, M. Di Francesco, and A. Passarella. Energy conservation in wireless sensor networks: A survey. *Ad Hoc Networks*, 7(3):537–568, 2009. 12, 14

[10] P. Andritsos, A. Fuxman, and R. J. Miller. Clean answers over dirty databases: A probabilistic approach. In *ICDE*, page 30, 2006. 4, 73, 96

[11] L. Antova, T. Jansen, C. Koch, and D. Olteanu. Fast and simple relational processing of uncertain data. In *ICDE*, 2008. 97

[12] A. Arion, H. Jeung, and K. Aberer. Efficiently maintaining distributed model-based views on real-time data streams. In *GLOBECOM*, pages 1–6, 2011. 13, 33

[13] M. Arulampalam, S. Maskell, N. Gordon, and T. Clapp. A tutorial on particle filters for online nonlinear/non-gaussian bayesian tracking. *IEEE Transactions on Signal Processing*, 50(2):174–188, 2002. 29, 30

[14] A. Bhattacharya, A. Meka, and A. Singh. MIST: Distributed indexing and querying in sensor networks using statistical models. In *VLDB*, pages 854–865, 2007. 3, 12, 31, 40, 105

[15] C. Bishop. *Pattern recognition and machine learning.* Springer, 2006. 122

[16] M. J. Bommarito II. Intraday Correlation Patterns between the S&P 500 and Sector Indices. *SSRN*, 2010. 44

[17] E. Brewer *et al.* . N-Smarts: Networked suite of mobile atmospheric real-time sensors. `http://www.cs.berkeley.edu/~honicky/nsmarts/`, 2007. 105

[18] J. Campbell, S. Grossman, and J. Wang. Trading volume and serial correlation in stock returns. *The Quarterly Journal of Economics*, 108(4):905, 1993. 44

[19] S. Cartier, S. Sathe, D. Chakraborty, and K. Aberer. ConDense: Managing data in community-driven mobile geosensor networks. In *IEEE SECON*, 2012. ii

[20] R. Cavallo and M. Pittarelli. The theory of probabilistic databases. In *VLDB*, pages 71–81, 1987. 96

[21] K. Chakrabarti, E. Keogh, S. Mehrotra, and M. Pazzani. Locally adaptive dimensionality reduction for indexing large time series databases. *ACM Transactions on Database Systems (TODS)*, 27(2):188–228, 2002. 2, 36

[22] K. Chan and W. Fu. Efficient time series matching by wavelets. In *ICDE*, pages 126–133, 1999. 13, 40, 41

[23] V. Chandola, A. Banerjee, and V. Kumar. Anomaly detection: A survey. *ACM Computing Surveys*, 41(3):1–58, 2009. 25

[24] R. Cheng, D. Kalashnikov, and S. Prabhakar. Evaluating probabilistic queries over imprecise data. In *SIGMOD*, pages 551–562, 2003. 2, 3, 30, 31, 73, 74, 79, 80, 96, 97, 107

[25] R. Cheng, D. Kalashnikov, and S. Prabhakar. Evaluation of probabilistic queries over imprecise data in constantly-evolving environments. *Information Systems*, 32(1):104–130, 2007. 2, 30

[26] R. Cheng, S. Singh, and S. Prabhakar. U-DBMS: A database system for managing constantly-evolving data. In *VLDB*, pages 1271–1274, 2005. 30, 97

[27] R. Cheng, Y. Xia, S. Prabhakar, R. Shah, and J. S. Vitter. Efficient indexing methods for probabilistic threshold queries over uncertain data. In *VLDB*, pages 876–887, 2004. 79, 80

[28] J. Chiles and P. Delfiner. *Geostatistics: modeling spatial uncertainty.* Wiley-Interscience, 1999. 102, 104, 109

[29] D. Chu, A. Deshpande, J. Hellerstein, and W. Hong. Approximate data collection in sensor networks using probabilistic models. In *ICDE*, pages 48–48, 2006. 2, 12, 14, 19, 40

[30] F. Chu, Y. Wang, S. Parker, and C. Zaniolo. Data cleaning using belief propagation. In *IQIS*, pages 99–104, 2005. 25

[31] R. Cole, D. Shasha, and X. Zhao. Fast window correlations over uncooperative time series. In *SIGKDD*, pages 743–749, 2005. i, 3, 71

[32] G. Cormode and M. Garofalakis. Sketching probabilistic data streams. In *SIGMOD*, pages 281–292, 2007. 3, 4, 73, 75, 96

[33] B. Costa *et al.* . Air Project. `http://www.pm-air.net/`, 2006. 105

[34] N. Dalvi and D. Suciu. Efficient query evaluation on probabilistic databases. *The VLDB Journal*, 16(4):523–544, 2007. 3, 73, 74, 96

[35] N. Dalvi and D. Suciu. Management of probabilistic data: foundations and challenges. In *PODS*, pages 1–12, 2007. 96

[36] A. Deligiannakis, Y. Kotidis, and N. Roussopoulos. Compressing historical information in sensor networks. In *SIGMOD*, pages 527–538, 2004. 37, 38

[37] A. Deligiannakis, V. Stoumpos, Y. Kotidis, V. Vassalos, and A. Delis. Outlier-aware data aggregation in sensor networks. In *ICDE*, pages 1448–1450, 2008. 25

[38] A. Deshpande, C. Guestrin, W. Hong, and S. Madden. Exploiting correlated attributes in acquisitional query processing. In *ICDE*, pages 143–154, 2005. 2, 12, 14, 16

[39] A. Deshpande, C. Guestrin, S. Madden, J. Hellerstein, and W. Hong. Model-driven data acquisition in sensor networks. In *VLDB*, pages 588–599, 2004. 2, 12, 14, 16, 17

[40] A. Deshpande and S. Madden. MauveDB: Supporting model-based user views in database systems. In *SIGMOD*, pages 73–84, 2006. 2, 3, 12, 14, 27, 40, 97, 102, 105, 107

[41] F. Diebold, T. Gunther, and A. Tay. Evaluating density forecasts with applications to financial risk management. *International Economic Review*, 39(4):863–883, 1998. 78

[42] R. Elmasri and S. Navathe. *Fundamentals of database systems.* Addison Wesley, 6th edition, 2010. 27, 28

[43] H. Elmeleegy, A. Elmagarmid, E. Cecchet, W. Aref, and W. Zwaenepoel. Online piece-wise linear approximation of numerical streams with precision guarantees. In *VLDB*, pages 145–156, 2009. 32, 33, 35

[44] E. Elnahrawy and B. Nath. Cleaning and querying noisy sensors. In *WSNA*, pages 78–87, 2003. 12, 25, 40

[45] M. Ester, H. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *SIGKDD*, pages 226–231, 1996. 110

[46] C. Faloutsos, M. Ranganathan, and Y. Manolopoulos. Fast subsequence matching in time-series databases. In *SIGMOD*, pages 419–429, 1994. 13, 39, 41, 71

[47] C. Franke and M. Gertz. ORDEN: Outlier region detection and exploration in sensor networks. In *SIGMOD*, pages 1075–1077, 2009. 12, 21, 25

[48] S. Gandhi, S. Nath, S. Suri, and J. Liu. GAMPS: Compressing multi sensor data by grouping and amplitude scaling. In *SIGMOD*, pages 771–784, 2009. 2, 13, 37, 71

[49] D. Ganesan, D. Estrin, and J. Heidemann. DIMENSIONS: Why do we need a new data handling architecture for sensor networks? In *SIGCOMM*, pages 143–148, 2003. 40

[50] D. Ganesan, B. Greenstein, D. Perelyubskiy, D. Estrin, and H. J. An evaluation of multi-resolution storage for sensor networks. In *SenSys*, pages 89–102, 2003. 40

[51] J. Gehrke, F. Korn, and D. Srivastava. On computing correlated aggregates over continual data streams. In *SIGMOD*, pages 13–24, 2001. 71

[52] G. Golub and C. Van Loan. *Matrix computations.* The Johns Hopkins University Press, 1996. 53, 114

[53] C. Guestrin, P. Bodik, R. Thibaux, M. Paskin, and S. Madden. Distributed regression: An efficient framework for modeling sensor network data. In *IPSN*, pages 1–10, 2004. 3, 12, 14, 105

[54] H. Gupta, V. Navda, S. Das, and V. Chowdhary. Efficient gathering of correlated data in sensor networks. *ACM Transactions on Sensor Networks (TOSN)*, 4(1):4, 2008. 12, 14

[55] R. Gupta and S. Sarawagi. Creating probabilistic databases from information extraction models. In *VLDB*, pages 965–976, 2006. 4, 73, 96

[56] O. Hassanzadeh and R. J. Miller. Creating probabilistic databases from duplicated data. *The VLDB Journal*, 18(5):1141–1166, 2009. 4, 73, 96

[57] M. Hua, J. Pei, W. Zhang, and X. Lin. Ranking queries on uncertain data: a probabilistic threshold approach. In *SIGMOD*, pages 673–686, 2008. 3, 73

[58] J. Hull. *Options, futures and other derivatives.* Prentice Hall, 2009. 44

[59] H. Jagadish, A. Mendelzon, and T. Milo. Similarity-based queries. In *PODS*, pages 36–45, 1995. 71

[60] A. Jain, E. Chang, and Y.-F. Wang. Adaptive stream resource management using Kalman Filters. In *SIGMOD*, pages 11–22, 2004. 13, 25, 29

[61] R. Jampani, F. Xu, M. Wu, L. L. Perez, C. Jermaine, and P. J. Haas. MCDB: A monte carlo approach to managing uncertain data. In *SIGMOD*, pages 687–700, 2008. 97

[62] S. Jeffery, G. Alonso, M. Franklin, W. Hong, and J. Widom. A pipelined framework for online cleaning of sensor data streams. In *ICDE*, page 140, 2006. 21

[63] S. Jeffery, G. Alonso, M. Franklin, W. Hong, and J. Widom. Declarative support for sensor data cleaning. In *Pervasive*, pages 83–100, 2006. 12, 21, 26, 40

[64] S. Jeffery, M. Garofalakis, and M. Franklin. Adaptive cleaning for RFID data streams. In *VLDB*, pages 163–174, 2006. 2, 12, 25, 26

[65] H. Jeung, H. Lu, S. Sathe, and M. L. Yiu. Managing Evolving Uncertainty in Trajectory Databases. *TKDE (under review).* 76, 123

[66] B. Kanagal and A. Deshpande. Online filtering, smoothing and probabilistic modeling of streaming data. In *ICDE*, pages 1160–1169, 2008. 2, 12, 13, 29, 40, 97

[67] B. Kanagal and A. Deshpande. Indexing correlated probabilistic databases. In *SIGMOD*, pages 455–468, 2009. 71, 96

[68] Y. Ke, J. Cheng, and W. Ng. Correlation search in graph databases. In *SIGKDD*, pages 390–399, 2007. 71

[69] E. Keogh, K. Chakrabarti, M. Pazzani, and S. Mehrotra. Dimensionality reduction for fast similarity search in large time series databases. *KAIS*, 3(3):263–286, 2001. 2, 71

[70] E. Keogh, S. Chu, D. Hart, and M. Pazzani. An online algorithm for segmenting time series. In *ICDM*, pages 289–296, 2001. 34, 35, 41

[71] E. Keogh and M. Pazzani. An enhanced representation of time series which allows fast and accurate classification, clustering and relevance feedback. In *SIGKDD*, pages 239–241, 1998. 32

[72] N. Khoussainova, M. Balazinska, and D. Suciu. Towards correcting input data errors probabilistically using integrity constraints. In *MobiDE*, pages 43–50, 2006. 96

[73] A. Klein. Incorporating quality aspects in sensor data streams. In *PIKM*, pages 77–84, 2007. 25

[74] A. Klein and W. Lehner. Representing data quality in sensor data streaming environments. *Journal of Data and Information Quality*, 1(2):1–28, 2009. 25

[75] Y. Kotidis. Snapshot queries: Towards data-centric sensor networks. In *ICDE*, pages 131–142, 2005. 20

[76] A. Krause *et al.* . Towards community sensing. In *IPSN*, 2008. 105

[77] L. V. S. Lakshmanan, N. Leone, R. Ross, and V. S. Subrahmanian. ProbView: A flexible probabilistic database system. *ACM TODS*, 22(3):419–469, 1997. 96

[78] I. Lazaridis and S. Mehrotra. Capturing sensor-generated time series with quality guarantees. In *ICDE*, pages 429–440, March 2003. 34, 38, 40, 41

[79] Y. Le Borgne, S. Santini, and G. Bontempi. Adaptive model selection for time series prediction in wireless sensor networks. *Signal Processing*, 87(12):3010–3020, 2007. 38, 41

[80] J. Letchner, C. Re, M. Balazinska, and M. Philipose. Access methods for Markovian streams. In *ICDE*, pages 246–257, 2009. 96

[81] C.-S. Li, P. S. Yu, and V. Castelli. Hierarchyscan: A hierarchical similarity search algorithm for databases of long sequences. In *ICDE*, pages 546–553, 1996. i, 3, 64, 70

[82] M. Li, D. Ganesan, and P. Shenoy. PRESTO: Feedback-driven data management in sensor networks. *IEEE/ACM Transactions on Networking (TON)*, 17(4):1256–1269, 2009. 2, 12, 14, 17, 18, 40

[83] X. Lian and L. Chen. Efficient similarity search over future stream time series. *TKDE*, 20(1):40–54, 2008. 2, 71

[84] S. Lin, V. Kalogeraki, D. Gunopulos, and S. Lonardi. Online information compression in sensor networks. In *IEEE International Conference on Communications*, 2006. 37, 38

[85] C. Liu, K. Wu, and P. J. An energy-efficient data collection framework for wireless sensor networks by exploiting spatio-temporal correlation. *TPDS*, 18(7), 2007. 101

[86] L. Luo *et al.* . Sharing and exploring sensor streams over geocentric interfaces. In *GIS*, 2008. 105

[87] S. Madden, M. Franklin, J. Hellerstein, and W. Hong. TAG: A tiny aggregation service for ad-hoc sensor networks. *ACM SIGOPS Operating Systems Review*, 36(SI):131–146, 2002. 2, 14

[88] S. Madden, M. Franklin, J. Hellerstein, and W. Hong. The design of an acquisitional query processor for sensor networks. In *SIGMOD*, pages 491–502, 2003. 2, 14, 40

[89] S. Madden, M. Franklin, J. Hellerstein, and W. Hong. TinyDB: An acquisitional query processing system for sensor networks. *TODS*, 30(1):122–173, 2005. 2, 14, 31, 40

[90] R. Maronna, R. Martin, and V. Yohai. *Robust statistics.* Wiley Series in Probability and Statistics, 2006. 45, 50

[91] C. Mayfield, J. Neville, and S. Prabhakar. ERACER: A database approach for statistical inference and data cleaning. In *SIGMOD*, pages 75–86, 2010. 2, 12, 25, 26, 40, 96

[92] C. Miller and L. Hively. A review of validation studies for the Gaussian plume atmospheric dispersion model. In *Journal of Nuclear Safety Vol*, volume 28, 2009. 104

[93] T. Minka. A comparison of numerical optimizers for logistic regression. 2007. 82

[94] M. Mokbel, X. Xiong, and W. Aref. SINA: Scalable incremental processing of continuous queries in spatio-temporal databases. In *SIGMOD*, page 634, 2004. 2

[95] M. Mokbel, X. Xiong, S. Hambrusch, S. Prabhakar, and M. Hammad. PLACE: A Query Processor for Handling Real-time Spatio-temporal Data Streams. In *VLDB*, 2004. 2

[96] M. Mokbel, X. Xiong, M. Hammad, and W. Aref. Continuous Query Processing of Spatio-Temporal Data Streams in PLACE. *Geoinformatica*, 9(4), 2005. 2

[97] A. Mueen, S. Nath, and J. Liu. Fast approximate correlation for massive time-series data. In *SIGMOD*, pages 171–182, 2010. i, 2, 3, 7, 64, 70

[98] S. Nittel. A survey of geosensor networks: advances in dynamic environmental monitoring. In *Sensors*, 2009. 101

[99] C. Olston, J. Jiang, and J. Widom. Adaptive filters for continuous queries over distributed data streams. In *SIGMOD*, pages 563–574, 2003. 34

[100] D. Olteanu, J. Huang, and C. Koch. SPROUT: Lazy vs. eager query plans for tuple-independent probabilistic databases. In *ICDE*, pages 640–651, 2009. 2, 3, 73, 96

[101] T. Palpanas, M. Vlachos, E. Keogh, D. Gunopulos, and W. Truppel. Online amnesic approximation of streaming time series. In *ICDE*, pages 339–349, 2004. 36, 41

[102] S. Papadimitriou, J. Sun, and C. Faloutsos. Streaming pattern discovery in multiple time-series. In *VLDB*, pages 697–708, 2005. 13, 71

[103] T. Papaioannou, M. Riahi, and K. Aberer. Towards online multi-model approximation of time series. In *IEEE MDM*, pages 33–38, 2011. 33, 38, 41

[104] A. Patcha and J.-M. Park. An overview of anomaly detection techniques: Existing solutions and latest technological trends. *Computer Networks*, 51(12):3448–3470, 2007. 25

[105] A. Petrosino and A. Staiano. A neuro-fuzzy approach for sensor network data cleaning. In *KES*, pages 140–147, 2007. 12, 25, 40

[106] D. Pollard. *A user's guide to measure theoretic probability.* Cambridge University Press, 2002. 88

[107] I. Popivanov. Similarity search over time series data using wavelets. In *ICDE*, pages 212–221, 2002. 2, 13, 40, 41

[108] D. Rafiei and A. Mendelzon. Similarity-based queries for time series data. In *SIGMOD*, pages 13–25, 1997. 2, 71

[109] J. Rao, S. Doraiswamy, H. Thakkar, and L. Colby. A deferred cleansing method for RFID data analytics. In *VLDB*, pages 175–186, 2006. 2, 12, 25, 26

[110] C. Ré, J. Letchner, M. Balazinksa, and D. Suciu. Event queries on correlated probabilistic streams. In *SIGMOD*, pages 715–728, 2008. 2, 3, 31, 71, 73, 96, 107

[111] G. Reeves, J. Liu, S. Nath, and F. Zhao. Managing massive time series streams with multi-scale compressed trickles. In *VLDB*, pages 97–108, 2009. i, 2, 3, 71

[112] H. Samet. *Foundations of multidimensional and metric data structures.* Morgan Kaufmann, 2006. 39

[113] S. Sathe and K. Aberer. AFFINITY: Efficiently querying statistical measures on time-series data. Technical report, EPFL, 2012. `http://infoscience.epfl.ch/record/180121`. i

[114] S. Sathe and K. Aberer. AFFINITY: Efficiently querying statistical measures on time-series data. In *ICDE (to appear)*, 2013. i, 7

[115] S. Sathe, S. Cartier, D. Chakraborty, and K. Aberer. Effectively modeling data from large-area community sensor networks. In *IPSN*, pages 95–96, 2012. ii

[116] S. Sathe, H. Jeung, and K. Aberer. Creating probabilistic databases from imprecise time-series data. In *ICDE*, pages 327–338, 2011. ii, 2, 4, 24, 107

[117] M. Sharaf, J. Beaver, A. Labrinidis, and P. Chrysanthis. TiNA: A scheme for temporal coherency-aware in-network aggregation. In *MobiDE*, pages 69–76, 2003. 2, 14

[118] W. Sharpe. Capital asset prices: A theory of market equilibrium under conditions of risk. *Journal of Finance*, 19(3):425–442, 1964. 44

[119] B. Sheng, Q. Li, W. Mao, and W. Jin. Outlier detection in sensor networks. In *MobiHoc*, pages 219–228, 2007. 25

[120] R. Shumway and D. Stoffer. *Time series analysis and its applications*. Springer-Verlag, New York, 2005. 18, 75, 80, 81, 82, 95

[121] A. Silberstein, R. Braynard, G. Filpus, G. Puggioni, A. Gelfand, K. Munagala, and J. Yang. Data-driven processing in sensor networks. In *CIDR*, 2007. 2, 20

[122] S. Subramaniam, T. Palpanas, D. Papadopoulos, V. Kalogeraki, and D. Gunopulos. Online outlier detection in sensor data using non-parametric models. In *VLDB*, pages 187–198, 2006. 2, 25

[123] Y. Tan, V. Sehgal, and H. Shahri. SensoClean: Handling noisy and incomplete data in sensor networks using modeling. Technical report, University of Maryland, 2005. 25, 40

[124] Y. Tao, R. Cheng, X. Xiao, W. K. Ngai, B. Kao, and S. Prabhakar. Indexing multi-dimensional uncertain data with arbitrary probability density functions. In *VLDB*, pages 922–933, 2005. 3, 73, 96

[125] A. Thiagarajan and S. Madden. Querying continuous functions in a database system. In *SIGMOD*, pages 791–804, 2008. 2, 12, 28, 33, 40, 105

[126] T. Tran, C. Sutton, R. Cocci, Y. Nie, Y. Diao, and P. Shenoy. Probabilistic inference over RFID streams in mobile environments. In *ICDE*, pages 1096–1107, 2009. 2, 4, 12, 25, 31, 40, 75, 96

[127] D. Tulone and S. Madden. PAQ: Time series forecasting for approximate query answering in sensor networks. In *EWSN*, pages 21–37, 2006. 12, 14, 80

[128] D. Wang, E. Michelakis, M. Garofalakis, and J. Hellerstein. Bayesstore: Managing large, uncertain data repositories with probabilistic graphical models. *PVLDB*, 1(1):340–351, 2008. 97

[129] L. Wang and A. Deshpande. Predictive modeling-based data collection in wireless sensor networks. In *EWSN*, pages 34–51, 2008. 13

[130] W. Willett *et al.* . Common sense community: scaffolding mobile sensing and analysis for novice users. In *Pervasive Computing*, 2010. 105

[131] E. Wu, Y. Diao, and S. Rizvi. High-performance complex event processing over streams. In *SIGMOD*, pages 407–418, 2006. 31

[132] H. Xiong, S. Shekhar, P. Tan, and V. Kumar. TAPER: A two-step approach for all-strong-pairs correlation query in large databases. *TKDE*, pages 493–508, 2006. i, 3, 71

[133] Z. Yan, V. Subbaraju, D. Chakraborty, A. Misra, and K. Aberer. Energy-efficient continuous activity recognition on mobile phones: an activity-adaptive approach. In *ISWC*, pages 17–24. IEEE, 2012. 3

[134] Y. Yao and J. Gehrke. Query processing in sensor networks. In *CIDR*, 2003. 2, 14

[135] B.-K. Yi, H. Jagadish, and C. Faloutsos. Efficient retrieval of similar time sequences under time warping. In *ICDE*, pages 201–208, 1998. 36

[136] Y. Zhang, N. Meratnia, and P. Havinga. Outlier detection techniques for wireless sensor networks: A survey. *IEEE Communications Survey & Tutorials*, 12(2), 2010. 21, 25

[137] Y. Zhu and D. Shasha. StatStream: Statistical monitoring of thousands of data streams in real time. In *VLDB*, pages 358–369, 2002. i, 3, 7, 13, 39, 64, 70

[138] Y. Zhuang, L. Chen, X. Wang, and X. Lian. A weighted moving average-based approach for cleaning sensor data. In *ICDCS*, page 38, 2007. 12, 21, 23, 40

Saket Sathe

Ph.D., EPFL, Switzerland

Rue du Villars 15
1024 Ecublens, Switzerland
📱 *+41 78 898 8759*
☎ *+41 21 693 1240*
✉ *saket.sathe@epfl.ch*
🖱 *saketsathe.net*

Expertise

Time-Series Data Mining, Mobile/Sensor Data Management, Time Series Analysis, Statistical Modeling, Probabilistic Databases, Distributed Computing.

Highlights

- 6+ years of experience in academic research and innovation.
- 1 year industrial experience as a software engineer.
- 10 papers in top conferences, like, ICDE, IPSN, IEEE SECON, etc.
- 1 US patent application filed.
- Supervised 5 masters students.

Education

2007–Present **Ph.D.**, *School of Computer and Communication Sciences, EPFL*, Switzerland.

- **Thesis Title : Statistical Models for Querying and Managing Time-Series Data.**
- Associated with the **Distributed Information Systems Laboratory**
- Advisor : Prof. Karl Aberer

2003–2006 **M.Tech.**, *Electrical Engineering, Indian Institute of Technology Bombay*.

- Thesis : Methods in Quantitative Risk Management
- Advisor : Prof. Uday Desai and Dr. Rajendra Lagu
- GPA : *9.11/10*

Work & Project Experience

Summer-2008 **Research Intern**, *Computational Research Laboratories*, Pune.

- Investigated use of GPUs for computational finance.

2007-2008 **Research Collaborator**, *OKKAM*, European Union Project.

- Architecture design and development of a distributed entity store (www.okkam.org).

2006-2007 **Software Engineer**, *GS Lab*, Pune.

- Worked on a Linux-based XML Security module and designing blog ranking algorithms.

2003-2006 **System Administrator**, *IIT Bombay*, Mumbai.

- Assisted in setting-up and managing the institute-wide LDAP database and campus DMZ.

Publications

Conferences, Workshops, Book Chapters

[1] A. Oviedo, **S. Sathe**, D. Chakraborty, and K. Aberer, "ENVIROMETER : A Platform for Querying Community-Sensed Data," in ***VLDB*** *(demo, submitted)*, 2013.

[2] **S. Sathe** and K. Aberer, "AFFINITY : Efficiently Querying Statistical Measures on Time-Series Data," in ***ICDE*** *(to appear)*, 2013.

[3] **S. Sathe**, T. Papaioannou, H. Jeung, and K. Aberer, ***Managing and Mining Sensor Data***. Springer, 2012, ch. A Survey of Model-based Sensor Data Acquisition and Management, ed. Charu Aggarwal.

[4] S. Cartier, **S. Sathe**, D. Chakraborty, and K. Aberer, "ConDense : Managing Data in Community-driven Mobile Geosensor Networks," in ***IEEE SECON***, 2012.

[5] **S. Sathe**, S. Cartier, D. Chakraborty, and K. Aberer, "Effectively Modeling Data from Large-area Community Sensor Networks," in ***IPSN***, 2012, pp. 95–96.

[6] **S. Sathe**, H. Jeung, and K. Aberer, "Creating Probabilistic Databases from Imprecise Time-Series Data," in ***ICDE***, 2011, pp. 327–338.

[7] K. Aberer, **S. Sathe**, D. Chakraborty, A. Martinoli, G. Barrenetxea, B. Faltings, and L. Thiele, "OpenSense : Open Community Driven Sensing of Environment," in ***ACM SIGSPATIAL IWGS***, 2010, pp. 39–42.

[8] H. Jeung, S. Sarni, I. Paparrizos, **S. Sathe**, K. Aberer, N. Dawes, T. Papaioannou, and M. Lehning, "Effective Metadata Management in Federated Sensor Networks," in ***IEEE SUTC***, 2010, pp. 107–114, (invited paper).

[9] E. Ioannou, **S. Sathe**, N. Bonvin, A. Jain, S. Bondalapati, G. Skobeltsyn, C. Niederée, and Z. Miklos, "Entity Search with NECESSITY," in ***SIGMOD WebDB***, 2009, (demo).

[10] **S. Sathe** and U. Desai, "Cell Phone Based Microcredit Risk Assessment using Fuzzy Clustering," in ***IEEE ICTD***, 2006, pp. 233–242.

[11] **S. Sathe**, "A Novel Bayesian Classifier using Copula Functions," 2006, arXiv :cs/0611150v3.

Journals

[12] H. Jeung, H. Lu, **S. Sathe**, and M. L. Yiu, "Managing Evolving Uncertainty in Trajectory Databases," ***TKDE*** *(submitted)*.

[13] I. Paparrizos, H. Jeung, S. Sarni, **S. Sathe**, and K. Aberer, "An Interactive System for Sensor Data Outlier Detection : End-to-End Solution with Model-based Approaches." *ACM Transactions on Interactive Intelligent Systems (submitted)*.

[14] **S. Sathe**, R. Lagu, and U. Desai, "Investigating Efficiency of the Indian Equities Market with Application to Risk Management." *Journal of Applied Finance*, vol. 12, no. 5, pp. 48–68, May 2006.

Technical Reports

[15] **S. Sathe** and K. Aberer, "AFFINITY : Efficiently Querying Statistical Measures on Time-Series Data," Tech. Rep., 2012, http://infoscience.epfl.ch/record/180121.

[16] S. Cartier, **S. Sathe**, D. Chakraborty, and K. Aberer, "ConDense : Managing Data in Community-driven Mobile Geosensor Networks," Tech. Rep., 2012, http://infoscience.epfl.ch/record/174752.

[17] A. Arion and **S. Sathe**, "Efficient Model-Driven Query Processing Based on Data Regeneration," Tech. Rep., 2009, https://infoscience.epfl.ch/record/178330.

[18] **S. Sathe**, "Rumor Spreading in LiveJournal," Tech. Rep., 2008, http://infoscience.epfl.ch/record/176326.

Patents

S. Sathe and G. Skobeltsyn : Method of Data Retrieval, and Search Engine using such a Method. EPFL. January 2011 : US 2011/0022600A1

Teaching Experience

2008-2010 **Teaching Assistant**, *Distributed Information Systems*, (in English).

Spring 2009 **Teaching Assistant**, *Programmation en Java (Java Programming)*, (in French).

2007-2012 **Project Supervisor**, *Supervised **four** semester projects and **one** masters thesis.*

Projects

2011-2013 **OpenSense**, *Open sensor network for air quality monitoring*.

Community-based wireless sensor network for monitoring air pollution.
- Proposed model-based techniques for query processing over community-sensed mobile geo-sensor data,
- Co-developed an application for obtaining on-the-fly pollution updates using smart phones,
- Full paper published in IEEE SECON 2012 and poster paper published in IPSN 2012,
- Funded by the SNSF Nano-tera initiative,
- Project URL : http ://www.nano-tera.ch/projects/401.php

2011-2013 **Swiss Experiment**, *Large-scale environmental monitoring using wireless sensor networks*.

A platform to enable real-time environmental experiments through wireless sensor networks.
- Researched methods for quantifying the uncertainty in sensor network data,
- Developed highly-efficient caching techniques for creating probabilistic databases from uncertain data,
- Full paper published in ICDE 2011,
- Project URL : http ://www.swiss-experiment.ch/

2007-2009 **OKKAM**, *Enabling the Web of Entities*.

A web-scale open service called Entity Name System (ENS) for supporting the systematic reuse of identifiers for “things”.
- Designed an entity (key-value) store and search engine using Apache Lucene and Hadoop,
- Project demo presented at WebDB 2009 (co-located with SIGMOD),
- Funded by the European Union FP7 Programme,
- Project URL : http ://www.okkam.org/

2007 **Pywebgraph**, *Power law graph generator*.

A threaded Power law random graph generator written in Python.
- **Open-source project with 3000+ downloads**
- Sole contributor to the project,
- Independently used by researchers from Microsoft Research and UCSB,
- Implements a threaded variant of the R-MAT algorithm for generating power law graphs,
- Computes strongly connected components,
- Project URL : http ://pywebgraph.sourceforge.net/

2011 **Conftrotter**, *Conference tracker and search engine*.
- Sole contributor to the project,
- Designed a web crawler for crawling conference CFPs posted on websites,
- Developed a fully-functional website for searching conferences using jQuery, Django and MySQL.