uOttawa

L'Université canadienne
Canada's university

FACULTÉ DES ÉTUDES SUPÉRIEURES ET POSTOCTORALES

uOttawa

L'Université canadienne
Canada's university

FACULTY OF GRADUATE AND POSDOCTORAL STUDIES

Samer Samarah

AUTEUR DE LA THÈSE / AUTHOR OF THESIS

Ph.D. (Computer Science)

GRADE / DEGREE

School of Information Technology and Engineering

FACULTÉ, ÉCOLE, DÉPARTEMENT / FACULTY, SCHOOL, DEPARTMENT

Knowledge Discovery for Behavioral Patterns in Wireless Sensor Networks

TITRE DE LA THÈSE / TITLE OF THESIS

Azzedine Boukerche

DIRECTEUR (DIRECTRICE) DE LA THÈSE / THESIS SUPERVISOR

CO-DIRECTEUR (CO-DIRECTRICE) DE LA THÈSE / THESIS CO-SUPERVISOR

EXAMINATEURS (EXAMINATRICES) DE LA THÈSE / THESIS EXAMINERS

Ferhat Khendek

Hussein Mouftah

Abed El Saddik

Dorina Petriu

Gary W. Slater

Le Doyen de la Faculté des études supérieures et postdoctorales / Dean of the Faculty of Graduate and Postdoctoral Studies

Knowledge Discovery for Behavioral Patterns in Wireless Sensor Networks

by

Samer Samarah

Thesis submitted to the
Faculty of Graduate and Postdoctoral Studies
In partial fulfillment of the requirements
For the Ph.D. degree in
Computer Science

School of Information Technology and Engineering
Faculty of Engineering
University of Ottawa

Your file Votre référence
ISBN: 978-0-494-48668-9
Our file Notre référence
ISBN: 978-0-494-48668-9

DEDICATION

To the memory of my Grandfather,

To my great Father and Mother,

To my Brothers and Sisters.

Samer

Abstract

The research consolidated in this thesis is motivated by the recent evolvement of wireless technologies and microelectronic devices, which instigated the emergence of Wireless Sensor Networks (WSNs). Many WSN-based applications have come about; these applications are related, but not limited, to the fields of military, environment and health care. Research in WSNs is still in its early stages, and efforts have been put forward to design fast, reliable, and fault-tolerant protocols that guarantee acceptable levels of quality for events delivery, to meet the limited capabilities of *sensor nodes* and the effects of unreliable wireless communication.

In this thesis, we focus on the design of a Knowledge-based framework for extracting behavioral patterns regarding sensor nodes from WSNs. Three types of behavioral patterns are introduced: Sensor Association Rules, Coverage-based Rules and Sensor Chronological Patterns. The proper steps in the Knowledge Discovery process that pertain to the extraction of the behavioral patterns are defined. These steps are: (i) a *formal definition* of the required 'knowledge'; (ii) the *data preparation* stage that covers the communication aspects of the process of preparing data that is needed to extract these patterns; (iii) the *data mining* techniques that are essential for extracting the required patterns. A set of schemes have been proposed to attain these steps, and meet the critical properties of WSNs. In contrast to other techniques, the proposed behavioral patterns are mainly about the sensor nodes, instead of the area under monitoring. The direct application of the proposed patterns is enhancing the performance of WSNs by participating in the resource management process of sensor nodes, and reducing the undesired cons of wireless communication; thus improving the *Quality of Service* of WSNs. Several experiments have been conducted, using synthetic and real data, to report about the performance of proposed schemes.

Acknowledgements

First and foremost, I offer all thanks and praise to Allah Almighty, for guiding and blessing me in this endeavor.

I am also greatly indebted to my supervisor, Prof. Azzedine Boukerche, for his invaluable help and guidance. This thesis would not have been possible without him.

My parents, brothers, sisters, and relatives offered me boundless support and encouragement, and to them I am forever grateful. I especially want to thank my uncle, Dr. Mohammed Al-Zoubi, and my brother-in-law, Dr. Ahmad AL-Zoubi. Ahmad, you are truly the "big brother" I always wanted.

A warm thank-you to all my friends in Ottawa, for the great times we have had together, and to my colleagues at PARADISE research lab, for their moral support. Special thanks to Ismial Shakre, Majdi Rawashdeh, and Muath Alzghool – Ismial, for proofreading my thesis, Majdi and Muath, for everything.

Finally, I wish to thank Yarmouk University – Jordan for their generous assistance.

Contents

List of Tables

List of Figures

Glossary of Terms

ASAP Adaptive Sampling APproach.

CADR Constrained Anisotropic Diffusion Routing.

CDMA Code Division Multiple Access.

CT Chronological Tree.

DKF Dual Kalman Filter.

DL Location Database.

DS Behavioral Database.

DSARM Data Stream Association Rule Mining.

KDD Knowledge Discovery in Database.

KDW Knowledge Discovery in Wireless sensor network.

LEACH Low-Energy Adaptive Clustering Hierarchy.

LMS Least Mean Square.

min_sup Minimum Support.

min_conf Minimum Confidence.

MNDGT Minimum Node Data Gathering Tree.

PAQ Probabilistic Adaptable Query.

PEGASIS Power Efficient GAthering in Sensor Information Systems.

PREMON PREdiction-based MONitoring.

PDA Personal Digital Assistant.

PLT Positional Lexicographic Tree.

PLT_P The Conditional Positional Lexicographic Tree of pattern P.

QoS Quality of Service.

SAR Sequential Assignment Routing.

SPIN Sensor Protocols for Information via Negation.

WASN Wireless Actor and Sensor Network.

WSN Wireless Sensor Network.

Chapter 1

Introduction

1.1 Background

Advances in wireless technologies and micro-electronic devices have led to the development of *sensor nodes* that are capable of sensing, processing and transmitting data [1, 81]. This new trend in sensor technology led the design of Wireless Sensor Networks (WSNs). A wireless sensor network consists of several sensor nodes that are designed to 'sense' the environment around them, and send, in cooperation with each other, detected events to a well-equipped node referred to as the Sink. The detected events are transmitted to the Sink in more than a way: periodically; upon satisfying a particular predicate (event-based); or as an answer to a query (query-based) [8]. Figure 1.1 shows the architecture of a simple WSN.

Wireless sensor networks have proven their success in a variety of applications, especially those that require fine-grained monitoring of physical environments that are subject to critical conditions, such as fire, toxic gas leaks and explosions [1, 81, 85]. In order to guarantee an acceptable level of quality for events' delivery, a new class of fast, reliable and fault tolerant protocols for WSN need to be developed [8]. However, the distributed nature and the limited resources of sensor nodes (e.g., energy, communication and computation [1, 81]), as well as the unreliability of wireless communication increase

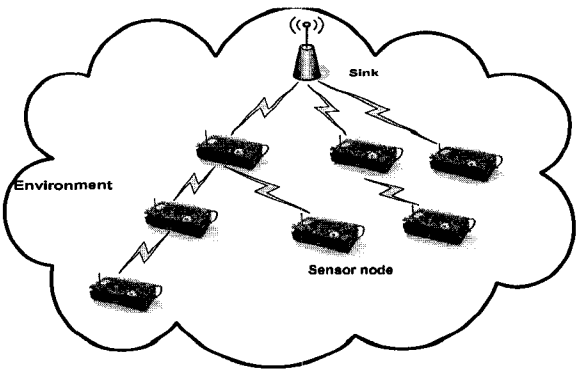


Figure 1.1: Wireless Sensor Network

the possibility of errors, lost messages, delays in data delivery and losses of functionality. These factors are potentially devastating to the performance and the overall Quality of Service (QoS) of WSNs [8, 38, 82, 90, 93].

Several techniques have been proposed in literature to enhance the performance and QoS of WSNs. These techniques have focused on two main aspects: The first is enhancing the resource usage of sensor nodes. Techniques in this direction range from clustering, multi-hop transmission, data aggregation and data fusion, just to mention a few [82, 94, 104, 100]. Although a combination of these techniques can be used in the same architecture, clustering and multi-hop transmission focus on reducing the distance that the data must travel, while data aggregation and fusion focus on reducing the amount of data to be sent to the Sink. The second aspect is devising solutions for tolerating latency, out of order event receipt and lost messages [38, 39, 78, 79].

Recently, *Knowledge Discovery Process*, a well known process in traditional database systems, used for extracting patterns from data [6, 5], has shown to be a promising tool for improving WSN performance and its quality of services [40]. *Knowledge Discovery in Wireless sensor network (KDW)* has is used to extract two types of information (knowledge): (i) Patterns about the surrounding environment, extracted from the data reported by sensor nodes [45, 46, 51, 99]. (ii) Behavioral patterns about sensor nodes, extracted from meta-data describing sensor behavior. These kinds of patterns are referred

to as sensor behavioral patterns.

The Knowledge Discovery process is by no means a trivial task; it requires a series of steps that include: Domain understanding, knowledge definition, data preparation and data mining [5, 6]. Knowledge Discovery in Wireless sensor networks (KDW) is not, as of yet, a well-defined process. The 'transformation' from Knowledge Discovery in Databases (KDD) to Knowledge Discovery in Wireless sensor networks (KDW) warrants careful planning and refining for most of the techniques in KDD. Moreover, new techniques that are designed specifically for the knowledge discovery process in WSNs need to be envisioned [40].

In this thesis, we propose a framework for extracting three types of sensor behavioral patterns, which we refer to as *Sensor Association Rules*, *Coverage-based Rules* and *Sensor Chronological Patterns.* The proposed framework defines the proper steps in the Knowledge Discovery process that pertain to the extraction of the behavioral patterns. These steps are: (i) a *formal definition* of the required knowledge (i.e., the behavioral patterns) (ii) the *data preparation* stage that covers the communication aspects of the process of preparing the data needed to extract these patterns (iii) the *data mining*techniques needed to extract the required patterns.

Data Mining is the field that is concerned with extracting hidden knowledge from vast amounts of data using techniques inspired by different disciplines, such as databases, machine learning, artificial inelegant and statistics [2, 3, 4]. Data mining is an essential step in the Knowledge Discovery process, and is a main component of this thesis.

The remainder of this chapter is organized as follow: Section 1.2 introduces real examples to illustrate the importance of the proposed behavioral patterns for WSNs, and presents the main challenges of generating these patterns. Section 1.3 presents the objectives and contributions of this research. Section 1.4 illustrates the wireless sensor network architecture that is used in this thesis. Section 1.5 presents the overall thesis organization.

1.2 Motivation and Research Challenges

Extracting behavioral patterns that describe sensor activities in wireless sensor networks is a complex process that requires extensive efforts. In that matter, several issues must be addressed: the characteristics of behavioral patterns, the effects of these patterns on wireless sensor networks, and the main challenges that face the Knowledge Discovery process when generating these patterns. In this section, we introduce the proposed behavioral patterns, Sensor Association Rules, Coverage- based Rules and the Chronological Patterns. We also provide some scenarios that motivate the generation of these patterns. Then, we list the main challenges that face the Knowledge Discovery process when extracting these patterns from wireless sensor networks.

Our definition of sensor behavioral patterns is based on two well known knowledge concepts in the domain of transactional databases: Association Rules and Sequential Patterns [11, 32]. These concepts have proven to be useful in driving the business process and helping managers in making decisions that affect their companies. At this stage, we will present the behavioral patterns from the wireless sensor network perspective; later we will describe the Association Rules and Sequential Patterns in more details.

Sensor Association Rules is the first behavioral pattern proposed in this thesis; it aims to capture the temporal relations between sensor nodes, based on common intervals of their activities. An example of sensor rule is $(s_1 s_2 \Rightarrow s_3, 90\%, \lambda)$, which translates: "if events from sensors s_1 and s_2 are received, then there is a 90% chance of receiving an event from sensor s_3 within λ units of time." Formulating Sensor Association Rules entails: finding patterns of sensors that detect events within the same time interval; and the frequency of the pattern. Frequency in this context refers to the number of times in which the same set of sensors detect events within the same time interval. The frequency is computed over a given historical period provided by the application. Patterns have frequency exceeding a certain threshold are called Frequent Association patterns. In our example, the rule $(s_1 s_2 \Rightarrow s_3)$ is generated from the pattern $(s_1 s_2 s_3)$.

The major impacts of Sensor Association Rules that benefit many applications are the ability of predicting the sources of future events, and the ability of identifying the sets of temporally correlated sensors. Forecasting the sources of future events can be seen in a variety of applications, such as predicting faulty nodes (e.g., we are expecting to receive an event from a certain node and that event does not arrive), or identifying the source of the next event in case of emergency preparedness class of applications. Identifying correlated sensors helps in compensating for the effects of unreliable wireless communication, such as missed readings. Also, it helps in the resource management process, such as deciding which nodes can be switched safely to a sleep mode without affecting the coverage of the network.

Coverage-based Association Rules is the second type of behavioral patterns that is proposed in this thesis. It assumes that the network is divided into a set of locations, each covered by k sensors. In k-coverage networks, sensor nodes covering the same area will share the same behaviors. Instead of generating the association between all the sensor nodes, Coverage-based Rules discovers the correlation among the locations in the network. Like Sensor Association Rules, the major application of Coverage-based Rules is to predict the location of future events.

The third type of behavioral patterns introduced in this thesis is the Chronological Patterns. Chronological Patterns takes the same structure as the Association Patterns, with major emphasis on the order of the sensors (that appear in the same pattern) in detecting the events. For instance, ($[s_1 s_3 s_6]$, λ, 90%) may be a Chronological Pattern that translates into: sensors s_1, s_3, and s_6 detected events in the time interval λ, and in the order s_1, s_3, and s_6 respectively, with 90% referring to the frequency of the pattern.

One of the interesting applications of Chronological Patterns is their use in compensating for the out of order reception of events in WSNs that is caused by the delay in transmission media, which in its self, is due to the delay caused by: competition for media access, delay caused by sending a message from node to node and its multiplication in case of multi-hops, and delay caused by changes in the topology [38, 39]. Many appli-

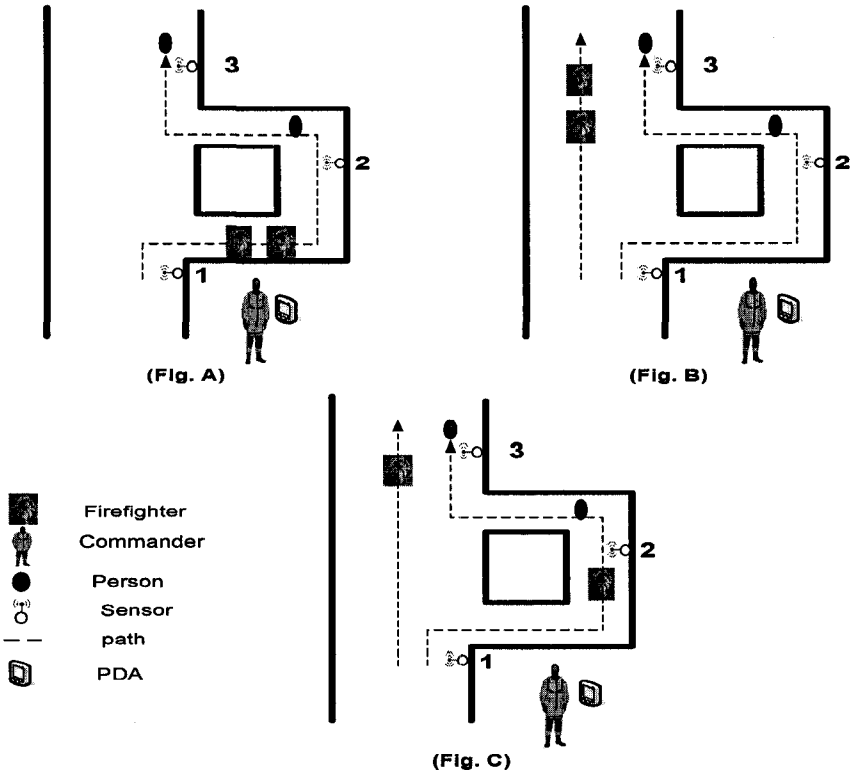


Figure 1.2: Event Ordering

cations require the exact ordering of events for correct interpretation. To illustrate the importance of event ordering, we introduce the following example. Here is an environment that is covered by a WSN with motion detection sensors. The WSN is interfaced through laptops or PDAs. Figure 1.2 shows a fire situation in which two persons are struggling to find an exit. A fire commander is trying to determine the exact location of the persons, in order to send a rescue team. Assume that the persons passed the locations covered by sensors s_1 and s_2. Assume that one person has successfully continued his way and passed the location covered by sensor s_3, and the other person has fallen after passing the location coversed by sensor s_2. Once a sensor detects a movement, it sends a notification message to the interface device. Figure 1.2-A shows the situation

in which the fire commander receives the events in the exact order (s_1, s_2, and s_3), and thus sends the rescue team to the location covered by sensor s_3 via the correct path. If these events are received out of order, let us say s_1 and s_3 (s_2 is not received yet due to delay caused by the transmission media), the result is incorrect interpretation of the right path, resulting in the commander sending the rescue team via the wrong path, as seen by Figure 1.2-B. To overcome this problem, WSNs should employ an event ordering mechanism that guarantees the exact ordering of the events, before delivering them to the application. However, running an ordering algorithm incurs extra delays in a situation in which time may be very critical. Our proposed solution is to reduce the effect of the out of ordering problem by providing the WSN with Chronological Patterns. The idea behind this is to start stream the events, as they arrive, to the application, with a degree of confidence pertaining to the exact ordering of the events, while an ordering mechanism is running in the background. In our example, if we know that the patterns $[s_1, s_3]$ and $[s_1, s_2, s_3]$ may occur with a chance of 40% and 80% respectively, then the team commander can send one member via the path $[s_1, s_3]$ and another member via the path $[s_1, s_2, s_3]$, as seen in Figure1.2-C.

Several challenges in discovering the proposed behavioral patterns from wireless sensor networks cast their shadow over the Knowledge Process, forcing a major modification on the traditional Knowledge Discovery process. These challenges include:

- The need for a formal definition of Sensor Rules, Coverage-based Rules and Chronological Patterns; definitions that identify the knowledge in terms of sensor terminologies.

- The need for data extraction mechanisms that are able to collect the data needed in the mining step, which takes into consideration the limited resources of sensor nodes, especially energy. It has to be emphasized at this point that the data involved in the Knowledge Discovery process is meta-data that describes sensor behaviors (which differs from data reported by the sensor about the surrounding

environment, i.e., sensors' readings). In this context, and throughout the entire thesis, when we refer to data extraction mechanisms, we are essentially referring to the techniques for preparing and collecting this meta-data.

- In most applications, wireless sensor networks are considered data stream systems that generate vast volumes of data in a short time. The stream nature reflects very active sources and thus a large amount of meta-data is generated. This entails the need for data structures that are capable of compressing and maintaining this amount of meat-data.

- To generate behavioral patterns, it is imperative to consider all possible relations that can be defined between sensors nodes. These relations must be checked against the meta-data to identify the relations of interest to the application. Sensor Association Rules and Coverage-based Rules are considered as exponational problem, while chronological pattern is considered as combinatorial problem. Efficient algorithms are needed by the mining step to generate these behavioral patterns in a reasonable mount of time and with minimal memory cost, in order to meet the critical class of applications.

1.3 Objectives and Contributions of Thesis

The main objective of this research is to address the challenges facing the Knowledge Discovery process in wireless sensor networks to extract Sensor Association Rules, Coverage-based Rules, and Chronological Patterns. To achieve our objective, we propose a comprehensive framework that defines the major steps for generating the proposed behavioral patterns in the Knowledge Discovery process. The knowledge steps considered in this thesis are knowledge definition, data preparation and data mining. Figure 1.3 gives an overview of the proposed framework, and summarizes the main contribution of this thesis. These are:

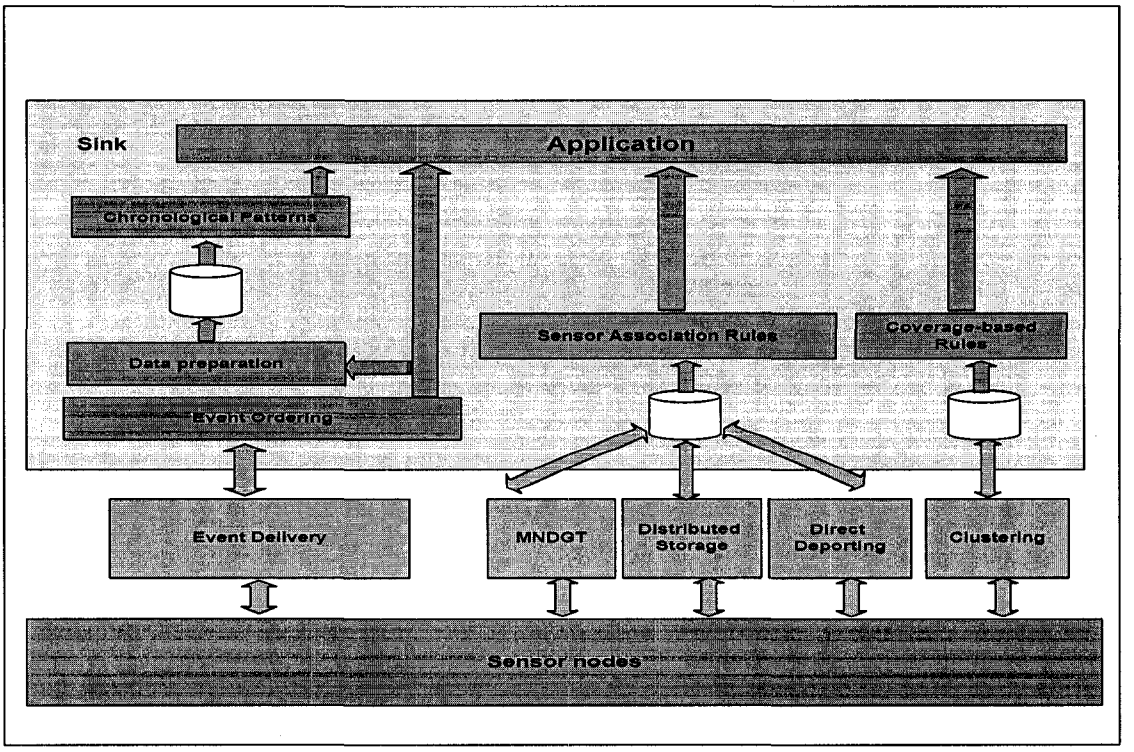


Figure 1.3: WSN Behavioral Patterns' Knowledge Discovery Process

- Three different types of sensor behavioral patterns:

  i *Sensor Association Rules*, which captures the temporal correlation between sensor activities.

  ii *Chronological Patterns*, which captures the ordered-temporal correlation between sensors activities.

  iii *Coverage-based Rules*, which captures the temporal correlation between a set of locations in k-coverage sensor networks.

- As part of the data preparation process for Sensor Association Rules, three different data extraction mechanisms are proposed in this thesis and evaluated; these are:

  i *Direct Reporting.*

  ii *Distributed Extraction.*

iii *In-network Data Reduction* mechanism, which is a built on top of a specifically designed routing tree named *Minimum Nodes Data Gathering Tree (MNDGT).*

These techniques are interfaced with sensor nodes, as illustrated in Figure 1.3; the collected data is stored in a database accessed by the Sink.

- Data preparation mechanism for collecting the data needed for mining Chronological Patterns. As shown in Figure 1.3, the proposed mechanism is designed on top of the event ordering mechanism that is employed by the network.

- To maintain the vast volumes of data generated for mining Sensor Association Rules and Chronological Patterns, two data structures maintained by the Sink, are proposed in this thesis for storing and compressing data:

  i *Positional Lexicographic Tree (PLT)* for storing activity data needed to generate the Sensor Association Patterns.

  ii *Chronological Tree (CT)* for storing data needed to generate the Chronological Patterns.

- To address the problems of high CPU time and memory size needed for generating the behavioral patterns, we propose two mining algorithms to generate Sensor Association Rules and Chronological Patterns from PLT and CT structures. The mining algorithms are implemented at the Sink node.

1.4 Network Architecture

The network architecture that acts as the playground for extracting data needed to generate the proposed behavioral patterns is assumed to consist of a set of sensor nodes and a well equipped node, known by the Sink. As shown in Figure 1.1. Sensor nodes use a multi-hop mode of transmission to route the data to the Sink. The Sink is attached

to a database to store the retrieved meta-data about sensor behaviors. Each node is assumed to be coupled with a flash memory device that stores mega bytes of data.

Attaching a storage device to each sensor was previously impossible due to the high energy consumption caused by maintaining the storage device. However, recent advances in flash memory technology have changed this perspective, and studies have shown that energy consumption for maintaining a unit of data within a flash memory coupled with a sensor node is very low compared to the energy needed for transmitting that unit. In [97], the authors reported that a flash memory like NANAD from Toshiba, storing 28GB of data generated at a rate of 512 bytes per second, will decrease the life time of a sensor node designed to run for three years by only six weeks.

1.5 Thesis Organization

This thesis consists of seven chapters, and is organized as follows: Chapter 2 reviews the main fields related to the thesis. Chapter 3 defines the Sensor Association Rules and presents three different extraction mechanisms to prepare the data needed in the mining process. Chapter 4 presents the Positional Lexicographic Tree, the data structure proposed to store data collected using the mechanisms addressed in Chapter three, and its mining algorithm. Chapter 5 introduces the Coverage-based Rules, which is designed specifically for k-coverage networks. Chapter 6 presents the Chronological Patterns, the data mechanism that is used to prepare data needed for mining these patterns, the Chronological Tree (the data structure proposed to store the prepared data), and the mining algorithm that generates the required patterns from the Chronological Tree. Chapter 7 concludes this thesis, and presents future works. For the convenience of the reader, we have presented the performance evaluation of each scheme in the chapter in which it appears.

Chapter 2

Literature Review

2.1 Introduction

The rapid development of Wireless Sensor Networks (WSNs) and their applicability to a wide range of applications [85, 87, 89, 91], has been motivating researchers to exploit their efforts, and develop methods and techniques, to improve the performance of WSNs. Among these methods, knowledge discovery process, has proved to be a successful tool for providing the sink of behavioral patterns pertaining to the sensor nodes that can be employed to enhance the performance of the WSN, and thus, its Quality of Service (QoS).

In order to discover the main challenges of the knowledge discovery process in WSNs, we explore the main factors affecting the integration of this process with WSNs, and check the state of the art implementations of that integration. In this chapter, we survey the main components of the knowledge discovery process in wireless sensor networks for behavioral patterns. We start by providing an overview of wireless sensor networks, their applications and their data delivery techniques. Then, we introduce the knowledge discovery process in traditional database systems, and briefly elaborate on the main role of each step of the discovery process, with special focus on the data mining step; we provide a detailed description of the data mining techniques that influence the behavioral

patterns proposed in this thesis. Finally, we introduce the knowledge discovery process from the wireless sensor network perspective, and survey the attempts that have been proposed in the literature, to integrate the knowledge discovery process with WSN, especially the data mining step.

2.2 Wireless Sensor Networks

Advances in wireless technologies and micro-electronic devices have led to the development of sensor nodes that are capable of sensing, processing, and transmitting data [81]. Sensor nodes are deployed in a certain geographical area, with the main functions of sensing the surrounding environment and sending the generated data to a central node, called the Sink, for further processing and analysis. Data can be sent directly to the Sink using the sensor nodes' radio transmitter; however, for many reasons, sensor nodes collaborate with each other to form a network, a Wireless Sensor Network (WSN), that relays the data to the Sink using a multi hop mode of transmission [1, 81, 90].

Several scenarios govern the interaction between sensor nodes and the Sink, usually defined by the application. These scenarios maybe periodic, event driven, or query based [8, 92]:

- In a periodic scenario, sensors send their readings to the Sink at regular intervals.
- In an event driven scenario, readings are sent to the Sink, once a particular event is detected.
- In a query based scenario, data is sent to the Sink, as an answer for a query, posted by the Sink.

Many WSN-based applications have emerged; these applications can be related, but not limited, to the fields of military, environment, and health care [8, 81, 85, 87, 88, 89, 90, 91]:

- **Military Applications**: Research on sensor networks was originally inspired by military applications. Examples of sensor networks for military applications range from battlefield surveillance and monitoring, intelligent missiles guidance systems, and mass destruction (such as chemical and nuclear) weapons attack detection. [85, 90].

- **Environmental Applications**: Sensors are deployed in natural environments, with the main goal of life monitoring, whether plant or animal life [91]; earthquake and flood detection, forest fire detection; toxic gas leakage, and explosions; population censuses [8, 81].

- **Health Care Applications**: Homes can be equipped with sensors that track patient behavior. The goal of such surveillance is to detect major incidents, such as falling off stairs or being inactive for a long period of time [87]. In another scenario, patients can be equipped with sensor devices that monitor heart activity, muscle dexterity, blood pressure, oxygen saturation, and other critical body functions. [88, 89].

Wireless sensor networks differ from traditional, wired and wireless, networks in many aspects. These differences are summarized as follows [1, 38, 82, 84]:

- Data flow in wireless sensor networks is many to one. Data travels from sensor nodes to the Sink, in contrast to other networks in which the communication can be between any two nodes.

- In many cases, data from different nodes in a WSN is concerned with a common subject, thus resulting in a high amount of redundancy.

- Sensor nodes have many limitations that restrict the design choices of WSN protocols. These limitations include limited processing, communications, and energy.

- WSNs are highly error-prone; they suffer from message loss and delay due to the unreliable wireless communications.

- WSNs lack a global addressing schema that governs the addressing process for sensor nodes.

Resource management is a key issue to be considered when designing WSN's protocols; it aims to optimize the resource usage in sensor nodes (communication, computation, and energy). In terms of energy consumption, communication and computation are the main tasks to be considered; they require great attention in WSN's protocols design process. On one end of the energy consumption gauge, protocols characterized by more computations and shorter transmission distances, are preferred over protocols that require more communications and that transmit over long distances; communication exhausts more energy than computation, and sending a unit of data to a node at distance r away, is more expensive than sending that unit of data to a closer node, [1, 94]. These facts have motivated developers to fathom several schemes, especially for routing techniques, that use multi-hop transmission, clustering, data aggregation, data fusion, and many other techniques that reduce the distance and the volumes of data needed to be sent to the Sink [82].

Routing is the most important aspect to consider when designing WSNs; it determines the success of the network. Routing protocols in WSNs have been classified into four different categories: data-centric, hierarchical, location-based, and quality of service-aware[82].

Data-centric protocols use a query-based approach in which a query diffuses from the Sink to the nodes hosting data that satisfies the query's predicates. Unlike what happens in address-centric protocols, used by wired and wireless ad hoc networks, in data-centric protocols the Sink asks for data regardless of its source. In address-centric protocols, however, a query is sent to nodes with specific addresses [90]. Several of the data-centric based protocols that have been proposed in the literature include, just to mention a few, Sensor Protocols for Information via Negation (SPIN) [95], Directed Diffusion [104], Rumor Routing [105], Energy Aware Routing [106], Gradient-Based Routing [107], and CADR (Constrained Anisotropic Diffusion Routing) [108]. SPIN was the

first routing protocol that used a data-centric approach, however, Directed Diffusion [104] is considered the algorithm that defined this category. Other algorithms, like those in [105, 106, 107, 108], are variations of the Directed Diffusion algorithm.

Hierarchical protocols feature sensor nodes that are organized into clusters, with each cluster having its own cluster head. Nodes within each cluster send their data to the cluster head, which in turn sends it to the Sink. Low-Energy Adaptive Clustering Hierarchy (LEACH) and Power Efficient GAthering in Sensor Information Systems (PEGASIS) are examples of the hierarchical-based approach [94, 109].

A location-based routing protocol uses information about a node's location, which can be obtained using GPS, to compute the distance between the source node and the Sink. The protocol then estimates the 'energy cost' of each possible path, eventually choosing the optimal one [1, 82]. Another important feature of this approach is its employment of location information; based on that information, a query is diffused to a particular area, instead of flooding the whole network with queries. Most of the location-based algorithms used in WSNs, have emerged from location-based algorithms designed for mobile ad hoc networks [82, 111, 112].

Quality of service-aware routing protocols consider certain quality metrics (such as delay, energy, bandwidth, etc.) in their routing decisions. Sequential Assignment Routing (SAR) [113] and SPEED [96] are examples of this kind of routing protocols.

The above classification is a brief taxonomy of WSNs' routing protocols. We believe, for the sake of abiding by the scope of this thesis, that it is better for the reader to refer to the survey presented in [82] for more details about any of the protocols. We will, however, delve shortly into more details on Directed Diffusion protocol in the next section, as it is one of the important routing protocols in WSNs. Furthermore; we discuss briefly the data gathering algorithms (a special case routing problem), to amplify on the characteristics of any algorithm used to prepare data needed for extracting behavioral patterns; which is an essential part of this thesis.

2.2.1 Directed Diffusion

Directed Diffusion, proposed by Intanagonwiwat *et al.,* [104], is the most popular data-centric routing protocol. It is a query-based routing protocol in which data is named using attribute-value pairs. The Sink initiates a query by broadcasting an interest to its neighboring sensor nodes. The interest, which may contain the attribute-value pair, data rate, region, time of initiation, and expiration time, describes the data that the Sinks wants from the sensor nodes. Upon receiving an interest, a node maintains it in its interest cache, where matched interests share the same entry. Each entry has a time stamp signifying the last matched interest. An entry also hosts several gradient fields, one a piece representing a neighboring node that sent an interest message; each gradient field consists of a data rate parameter, a duration parameter, and an expiration time parameter. Naturally, these values had been contained in the interest message itself, and were derived from the latter by the receiving node. The nodes then keep re-broadcasting the interest until the interest diffuses into the whole network. Once an event is detected by a sensor node, the node checks its cached interest entries for a match of the detected event, and if so, chooses the entry that exhibits the maximum data rate among the other interest gradient fields; it then generates the event, based on the chosen data rate. The event is then sent to each neighboring node. Finally, the Sink receives the event from different paths, and forces one of these paths by increasing its gradient.

2.2.2 Data Gathering Protocols

Data gathering is a special case routing problem, which assumes rounds of communications between sensor nodes and the Sink. In each round, data is collected from all sensors in the network [109]. What distinguishes data gathering protocols from other data routing protocols, is the high possibility of data aggregation and fusion. The goal of the gathering algorithms is to maximize the number of rounds in which data can be collected, with minimum delay and energy consumption. $Energy \times delay$ is the main

metric used to compare the performance of the data gathering algorithms [90, 110].

The direct and simple solution for the data gathering problem is to have each sensor node send its data directly to the Sink. However, this solution is very expensive in terms of energy consumption and delay [90, 110]; LEACH [94], PEGASIS [109], chain based binary approach [110], and chain-based three level [110] are more sophisticated schemes proposed for the data gathering problem in WSNs. Following is a brief description of each of these schemes.

LEACH (Low-Energy Adaptive Clustering Hierarchy) is a clustering-based routing protocol, proposed by Heinzelman *et al.,* [94]. LEACH follows the hierarchy approach, in which the network is divided into clusters, with each cluster having its own cluster head. Nodes within the cluster send their data to the cluster head, which in turn sends the data to the Sink. LEACH distributes the energy consumption between sensor nodes by randomly rotating the cluster head among the sensor nodes. Furthermore, LEACH optimizes the volume of data to be sent to the Sink by performing local data aggregation and fusion at the cluster head.

PEGASIS (Power-Efficient GAthering in Sensor Information Systems) is a data-gathering algorithm, proposed by Lindsey *et al.,* [109]. PEGASIS assumes that the nodes are homogenous; that those nodes possess limited energy; that the Sink is fixed and far away from the network; and that each node can transmit directly to the Sink. PEGASIS algorithm begins by forming a chain among the sensor nodes. For a given round, there is one leader among the nodes, and that leader will send the data to the Sink. Nodes take turns in being leader of the chain at different rounds. Each node receives data from a neighboring node, adds it to its own data, and transmits the accumulated data to its other neighbor in the chain. This process continues until all data is accumulated at the leader node, which sends it to the Sink. This data-gathering process starts at the last node in the chain. A token passing mechanism is employed to control nodal transmission. The token is initiated at the leader node, then passed to the last node in the chain. Figure 2.1 (the figure is redrawn from [109]) shows an example involving 5

nodes. c_2 is the leader node that sends the token to c_0. c_0 sends its data to c_1. c_1 adds the received data to its own and passes it to c_2. c_2 passes the token to c_4 and waits until it receives the data from c_3, accumulates all data, and finally sends it to the Sink.

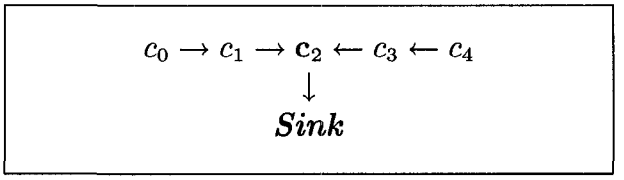


Figure 2.1: Token Passing in PEGASIS

Two factors deem PEGASIS victorious over LEACH in the energy saving duel: The short distances of message transmission, and the smaller volume of data received by the leader node [109]. However, PEGASIS performs poorly in terms of *Energy* $\times$ *Delay* metric; since each node must wait until it receives the token before it can transmit. Two variations of the PEGASIS protocol have been proposed in [110], to improve *Energy* $\times$ *Delay* metric, in CDMA (Code Division Multiple Access) sensor nodes and non CDMA sensor nodes.

In the **Chain-Based Binary Approach** using CDMA-capable sensor nodes, the network is divided into log(N) levels, N being the number of nodes. Nodes at the same level communicate pair-wise, and in parallel. A node that is a receiver at a level will be active at the next level. The node that remains at the end of the hierarchy will transmit the accumulated data to the Sink. Figurer 2.2 (the figure is redrawn from [110]) depicts an example of the chain based binary approach, in an eight-node network.

The Chain-Based Three Level approach is another variation of the PEGASIS protocol, proposed in [110], aiming to improve the *Energy* $\times$ *Delay* metric in the PEGASIS algorithm for the non CDMA sensor nodes. True to its name, this approach accomplishes data gathering over three levels. At the first level, nodes are divided into G groups. Member nodes within each group constitute a linear chain, with one node taking the role of chain leader. Each node transmits its data (one node at time) to the leader node. The leader then accumulates the data, and becomes a member node in another

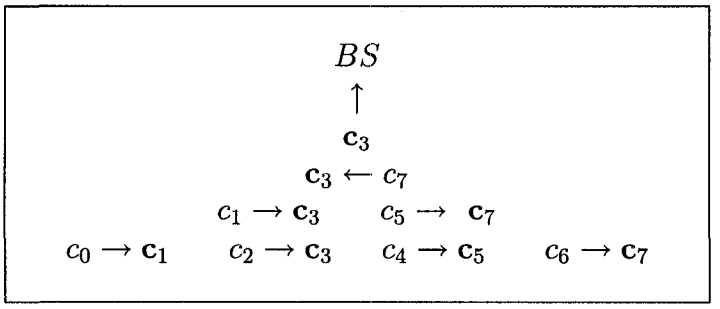


Figure 2.2: Chain-based Binary Scheme

group at the second level. At the second level, the N/G nodes (that were leader nodes at the previous level), N being the total number of nodes on the network, are divided into two groups. Repeating the data gathering scenario that had occurred at the previous level, data in each group is accumulated at the leader, which participates in the group of the two remaining nodes at the third, and final, level. After data is accumulated at one node, it is finally sent to the Sink. Figure 2.3 (the Figure is redrawn from [110]) illustrates a three level chain-based approach for N = 12 nodes and G = 3 groups. For equal energy distribution, nodes alternate in assuming the leader role in different rounds.

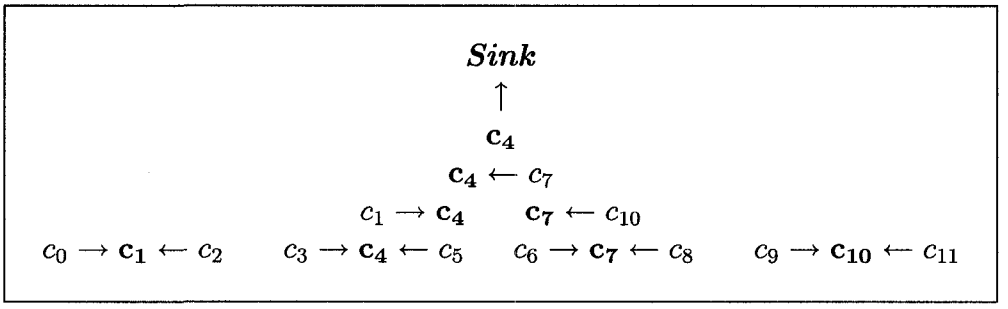


Figure 2.3: Chain-based Three Level Scheme

2.3 Knowledge Discovery and Data Mining

Knowledge Discovery in Databases (KDD) has received a great deal of attention, as a field that automates the process of analyzing vast volumes of data [6]. The primary goal of the

knowledge process is extracting hidden patterns, through a series of well defined steps, from a large volume of data [2]. The extracted patterns are vital to gaining information about the domain of data. Data mining is the main step in the knowledge discovery process, defining the purpose of the process as a whole. In this section, we define the knowledge discovery process in databases, identify the steps in the knowledge discovery process, and provide a detailed description of the data mining step and its techniques.

Data mining and KDD have inaccurately, in our opinion, been viewed as one and the same. Fayyad *et., al* [5] defines KDD as " *the non-trivial process of identifying valid, novel, potentially useful, and ultimately understandable patterns in data.*" Fayyad also defines data mining, labeling it as " *a step in the KDD process consisting of particular data mining algorithms that, under some acceptable computational efficiency limitations, produce a particular enumeration of patterns over the data.*".

The term, ***process***, in the KDD definition refers to iterative multi-steps of domain understanding, data selection, data preparation, data mining, and pattern evaluation and presentation. These steps are summarized from [5, 6]:

1. **Domain Understanding:** The goal of this step is to achieve a sound understanding of the application domain, prepare the prior knowledge, and identify the goal of the knowledge discovery process.

2. **Data Selection:** Data needed by the discovery process is identified. A sampling technique may be used to select a representative subset of the original data.

3. **Preparation:** This step involves 'treatment' of noise and missing values in the data; which could be intergraded from several sources, and a reduction process may be performed on it.

4. **Data Mining:** The data mining method that can extract the desired knowledge is identified and applied on the data to generate the required patterns.

5. **Pattern interpretation and presentation:** This step encompasses the inter-

pretation of the patterns, in order to present them in a manner that will facilitate their usage; visualization techniques may be used in this step.

The data mining step in the KDD process involves the choice of the appropriate data mining task, selection of the mining algorithms that will be used for generating the patterns, and the actual implementation of the mining algorithms [7]. Data mining tasks can vary depending on the type of patterns. These tasks include: classification, clustering, prediction, and association; they are summarized from [2, 3, 4, 5, 7, 10].

- **Classification:** Builds a model that describes a predefined set of classes, by analyzing a database of records containing attributes that describe these classes.
- **Regression:** Assigns a real value (continuous value) to a data item.
- **Time Series Analysis:** Examines the value of the attribute as it varies over time, to discover any correlation between the values of the attribute.
- **Association Rules:** Discovers relationships among objects that appear within the same context in a given database. Context is an application-dependent concept.
- **Sequence Discovery:** Infers sequential patterns in data that resulted from applications that had involved sequences of actions governed by time.
- **Summarization:** Concludes compact information representing a large data set; this process is similar to computing the tradition mean and variance in statistics.
- **Clustering:** Groups similar objects into classes. In contrast to the classification process, clustering is an unsupervised learning process.

The different works in literature look at classification, regression, and prediction from different angles. Below are the different perspectives:

1. Prediction is a separate task which refers the to the process of predicting the *future* data states, based on the current data [7]; classification, on the other hand, is used

to predict the *current* nominal values; and regression is used to predict the *current* continuous values.

2. Classification and regression are the two major types of prediction, where classification is used to predict nominal or discrete values, and regression is used to predict continuous or order values [2].

3. Classification is the process of predicting nominal classes, while prediction is the process of predicting continuous values (using regression techniques) [2].

In this thesis, we will adopt the second view that considers classification and regression as two types of prediction.

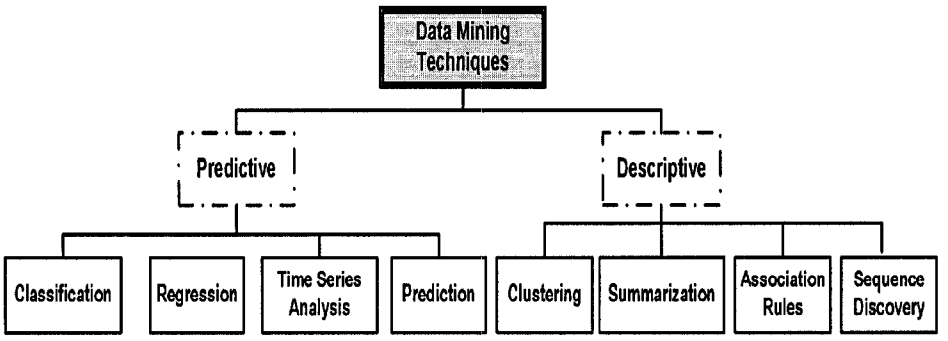


Figure 2.4: Data Mining Models and Tasks [7]

The goal of the data mining step is to create a model that fits the data on hand. Data mining models can be classified into predictive and descriptive models [7]. A predictive model is used to predict the values of unknown parameters based on historical data, while a descriptive model is used to extract patterns and relationships to describe the data. Figure 2.4 classifies the data mining tasks into predictive and descriptive models(the Figure is redrawn from [7]).

Data mining tasks can be achieved by applying techniques and algorithms inspired by different disciplines such as databases, machine learning, artificial intelligence, infor-

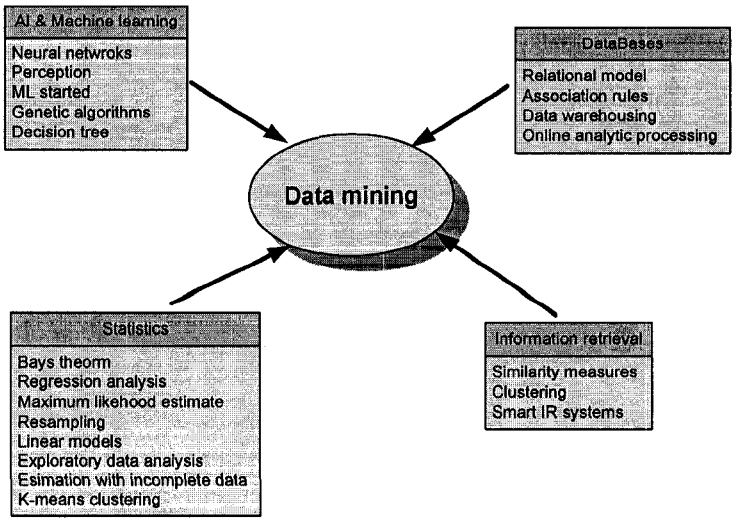


Figure 2.5: Data Mining Algorithms

mation retrieval, and statistics [2, 3, 4]. Figure 2.5 shows some of the fields and their algorithms, that influence some data mining tasks. [7].

The data mining tasks for the behavioral patterns proposed in this thesis are classified under association rules and sequence discovery. In the following two sections we present these tasks in more details.

2.3.1 Association Rules Algorithms

Association Rules algorithm was first introduced by Argawal *et al.*, [11] in 1993 to discover the correlations between objects in transactional databases, that appear in the same context. Market basket analysis is the main application of association rules; it uses them to predict purchase patterns of customers. Formulating predictions of such patterns help managers in making decisions about their businesses, such as appropriate items to go on sale, or placement of related items together (shelf management).

Association Rules is a data mining task that falls under two labels: predictive, and descriptive. A database that consists of a set of tuples, each containing a set of objects that

belong to the same context. This context can be items that are purchased together [11], or a set of symptoms that are characteristic of a particular disease [13]. Association Rules searches the database for patterns that best describes the relationship between the different objects. The searching process is based on some measurement provided by the application. In this section, we will introduce the association task in more details, briefly narrating its history, providing its formal definition, and listing the major techniques and algorithms that have been proposed to extract the 'Rules-in-Associating Rules' from a database. In order to seamlessly establishing the link between Association Rules and behavioral patterns proposed in this thesis, our presentation of Association Rules will be put about in an abstract manner, without refereing to a particular application.

The problem of mining association rules was formally defined in [12]as follows: Let $O = \{o_1, o_2, ..., o_n\}$ be a set of distinct objects; let D be a set of contexts where each context C is a set of objects such that $C \subseteq O$; each context has an associated identifier, uniquely identifying it, called CID. Table 2.1 shows an example of database of three contexts.

CID	Pattern
1	$o_1o_2o_3o_4$
2	$o_1o_3o_4$
3	$o_2o_4o_6$

Table 2.1: Database of Contexts

Let X be a subset of objects, such that $X \subseteq O$. X is called a pattern of objects, and a subset with K objects is called K-pattern. A context C supports X; if $X \subseteq C$. X is said to have a support s, if s of the contexts in D support X. For instance, in Table 2.1, the pattern o_1o_3 has support equal to 2 (i.e., it is supported by contexts 1 and 3).

The association rules are implications of the form $X \Rightarrow Y$, such that $X \subset O$, $Y \subset O$, and $X \cap Y = \phi$. The support of the rule $X \Rightarrow Y$ is the support of the pattern $X \cup Y$. The confidence of the rule is defined to be the fraction of the number of contexts in the database that support the patterns $(X \cup Y)$, to the number of contexts that support Y

(i.e., Support($X \cup Y$) / Support(Y)) .

The Association rules mining problem over database D, is finding all the association rules that have support and confidence greater than, or equal to, the user's predefined minimum support and minimum confidence percentage, respectively. Patterns that have support exceeding a given minimum support are called frequent patterns.

Based on Agrawal *et al.,* [11], the problem of mining association rules can be broken down into the following two tasks:

1. For each frequent pattern X, generate all that pattern's subsets.
2. For each subset Y of X, generate the rule $Y \Rightarrow (X - Y)$ if the subset's confidence is greater than, or equal to, the user's given minimum confidence.

Generating association rules is a straightforward process; all the algorithms proposed in the literature focus on how to generate the frequent pattern efficiently.

Generating frequent patterns is a costly problem that grows exponentially with the number of objects in the database. Theoretically, if we have a set of $\boldsymbol{n}$ objects, then there are 2^n potential relations (possible patterns) that can exist among these objects; these relations are used to formulate the association rules. To compute the support of these rules, the set of possible patterns should be checked against the database, in order to determine their supports. For large $\boldsymbol{n}$, the huge search space will degrade the performance of any algorithm designed to solve this problem. The computational costs can be further increased by the large size of a given database.

Generating frequent patterns is one of the problems that motivate researchers inventing new techniques and data representation models that can enhance the performance of the algorithms that were originally designed to solve this problem. Two main layout structures have been used for storing the database: horizontal and vertical. In the horizontal layout [11, 12], data is arranged in tuples of two attributes, context-id and pattern (i.e. the object that appear together in the same context). In the vertical layout [18, 19, 20], each object is listed, along with the identifers of the context in which it

exits. Tables 2.2 and 2.3 show examples of these structures.

Table 2.2: Horizontal Layout

CID	Pattern
1	$o_1o_2o_3o_4o_5o_6$
2	$o_1o_3o_4$
3	$o_2o_4o_6$
4	$o_1o_3o_5$
5	o_1o_4
6	$o_1o_2o_3o_4$

Table 2.3: Vertical Layout

Object	Context
o_1	1,2,4,6
o_2	1,3,6
o_3	1,2,4,5,6
o_4	1,2,3,5,6
o_5	1,4
o_6	1

Several algorithms have been proposed for generating the set of frequent patterns. These algorithms are classified into two main approaches: the candidate generation approach [11], and the pattern growth approach [24]. We present these approaches in the following sections.

2.3.1.1 Candidate Generation Approach

Following several scans of the database, the candidate generation approach enumerates the frequent patterns in a level-wise manner. In each iteration, a set of candidate patterns is generated and checked against the database, to determine the set of frequent patterns. This approach features such algorithms as: AIS [11], Apriori, AprioriId and AprioriHyprid [12], DHP (Directed Hashing and Pruning) [21], the Partition algorithm [23], and DIC (Dynamic Itemset Counting) [22]. Apriori is considered the core algorithm for this approach; it will described in more details below. The other algorithms, except AIS, are optimizations of the Apriori algorithm.

AIS algorithm (Agrawal, Imielinski, and Swami) was first proposed in 1993 [11]. It generates the candidate set on the fly (i.e., the candidate set is generated as the database is scanned); it makes several database scans. In any given scan, each context in the database is checked against the frequent patterns that were found in the pervious scan. Once a frequent pattern is found, it is extended by the frequent objects found in the context (one object at a time), and inserted into the candidate set. If the new

candidate already exists in the candidate set, its support will be incremented by one, otherwise it will be added, with a support count equal to 1, and will be considered later in the same pass. At the end of the pass, the candidate patterns with frequency less than minimum support are eliminated, and the remaining patterns are considered as frequent patterns and will be checked for the next pass.

The Apriori algorithm is considered the core algorithm for the candidate generation approach [12]. It assumes a horizontal layout for the database, and uses an anti-monotone property called *Apriori property*, stated below, to reduce the size of the search space.

Property 2.3.1. *All the non empty subsets of a frequent pattern are also frequent.*

The following steps illustrate the Apriori algorithm, using the database presented in Table 2.2, as a running example, with minimum support equal to 3.

1. Scan the database to identify the set of frequent one object (F_1). In our example, $F_1 = \{(o_1, 5), ((o_2, 3), (o_3, 4), (o_4, 5)\}$, the number beside each object refers to its frequency. Note that objects o_5 and o_6 are not frequent.

2. Generate the candidate set of the 2-patterns (C_2). C_2 is generated by self joining the object in the set F_1. In general, the candidate set C_k is generated by self joining the objects in the set F_{k-1}. To illustrate this process, let us refer to the j^{th} object in a pattern p, such that $P \in F_{k-1}$, by *p[j]*. The candidate set C_k is generated by the following SQ query [12]:

   Insert into C_k
   Select p[1],p[2],.....p[k-1],q[k-1]
   From F_{k-1} *p*, F_{k-1} *q*
   Where p[1] = q[1], ...,p[k-2] = q[k-2], p[k-1] < q[k-1].

   In our example, $C_2 = \{o_1o_2, o_1o_3, o_1o_4, o_2o_3, o_2o_4, o_3o_4\}$

3. Remove from the candidate set all the elements that have, at least, one infrequent subset (property 2.3.1). Since all the subsets of C_2 are frequent, non of them is removed.

4. Scan the database to determine the frequent set of 2-pattern (i.e., F_2) in the candidate set C_2. F_2 = { (o_1o_3, 4), (o_1o_4, 4), (o_2o_4, 3), (o_3o_4, 3) }.

5. Generate the candidate set of 3-patterns from the set F_2 using the above described SQL query. $C_3 = \{o_1o_3o_4\}$.

6. Scan the database to determine the set of frequent elements in C_3. In the above example, we have $F_3 = \{(o_1o_3o_4, 3)\}$.

7. At this stage, there is no candidate set that can be generated from F_3, thus the algorithm terminates.

In general, the candidate set C_k is formulated from the frequent set F_{k-1} using the SQL query described in step 2. The candidate set is pruned, and a database scan is conducted to identify the set F_k. The process is repeated until no candidate sets can be formulated. Table 2.4 shows the frequent patterns for the database in Table 2.2. Table 2.5 shows some of the association rules, with minimum confidence of 75%.

Table 2.4: Frequent Patterns

Pattern	**Support**
o_1	5
o_2	3
o_3	4
o_4	5
o_1o_3	4
o_1o_4	4
o_2o_4	3
o_3o_4	3
$o_1o_3o_4$	3

Table 2.5: Association Rules

Rule	**Support**	**Confidence**
$o_1 \Rightarrow o_3$	4	80%
$o_1 \Rightarrow o_4$	4	80%
$o_2 \Rightarrow o_4$	3	100%
$o_3 \Rightarrow o_4$	3	100%
$o_3 \Rightarrow o_1o_4$	3	75%
$o_1o_3 \Rightarrow o_4$	3	75%
$o_1o_4 \Rightarrow o_3$	3	75%

2.3.1.2 Pattern Growth Approach

Candidate generation algorithms have proven their efficiency in generating the frequent patterns, alas, they suffer from two main drawbacks [2]: First, a large number of candidates must be generated, stored, and maintained in order to determine their frequencies. Second, a large number of multiple scans of the database is needed for generating candidates' frequencies.

The pattern growth approach, which was first introduced by Han *et al.*, [24], has been proposed to overcome the limitations of the candidate generation approach. The main idea behind the pattern growth approach is to formulate long and frequent patterns from short ones, and use Projected Databases to compute patterns frequencies. Projected Database is an important concept; it restricts the search space of a pattern to a portion of the original database [15]. There are two essential steps that comprise the pattern growth approach. The first step realizes the projected database by building the FP-tree structure; the second step involves applying the FP-growth routine. The FP-tree is a prefix tree structure built for the purpose of compressing the database. It consists of a root node labeled "null", a set of prefix subtrees (as children of the root), and a header table that stores the frequent objects with their frequency values. Each node in the tree consists of a label; a counter, which is an integer value referring to the number of contexts, in the database, that share the prefix path from the root node to the node in question; and a node-link that connects the node in question to the next node, with the same label, in the tree. FP-growth is the mining routine that generates the set of all frequent patterns from the FP-tree, in a divide and conquer manner, following the pattern growth approach. The following steps illustrate the pattern growth method through a running example, using the database presented in Table 2.2.

1. The FP-tree must be constructed: a scan of the database is performed, to determine the set of frequent objects. The objects are then sorted in descending order of their frequencies, before they are stored in the header table. For the database in

Table 2.2, F_1= {(o_1,5), ((o_4,5),(o_3,4),(o_2,3)}.

2. Now that frequent objects in the database have been sorted, another scan is conducted, and the header table created in step 1 will be used to eliminate infrequent objects in each context in the database. The frequent objects in each context are sorted, and a branch comprising of nodes of these objects is created and inserted the FP-tree, starting from the root node. If several contexts share the same prefix ($o_1 \rightarrow o_4 \rightarrow o_3$ is an example of a prefix), the nodes count in each of the nodes in the prefix is incremented by one; otherwise, the count is induced with a support count equal to one. After a node's first occurrence during the database scan, that node is 'tied up' to the node-link of the entry corresponding to its label in the header table; on the occasion of another occurrence of a node, it is linked to the last node, with the same label, inserted in the tree. Taking the first context in the database of Table 2.2 (after removing the infrequent objects and sorting them according to the objects in the header table), for example, the branch ($o_1 \rightarrow o_4 \rightarrow o_3 \rightarrow o_2$) is created and inserted in the FP-tree with counts equal 1 for each node. Each node is linked to its image in the header table. For the second context in the table, the prefix ($o_1 \rightarrow o_4 \rightarrow o_3$) is created; observing that it shares a part of an already existing prefix, the nodes count of each shared node is incremented by 1. The same process is replayed for the other contexts in the database. Figure 2.6-A shows the FP-tree of the database presented in Table 2.2.

After constructing the FP-tree, the mining routine that will generate the set of frequent patterns is invoked, to start the mining process. The process's starting point is the object with the least frequency in the header table, say o_n. The pattern o_n is generated as the first frequent pattern of length one. After that, o_n's conditional sub patterns are extracted from the original FP-tree. The o_n's conditional sub patterns - branches in the FP-tree that extend from the root node to the nodes labeled o_n - can be extracted by following the node links, starting from node o_n's node-link in the header table; the

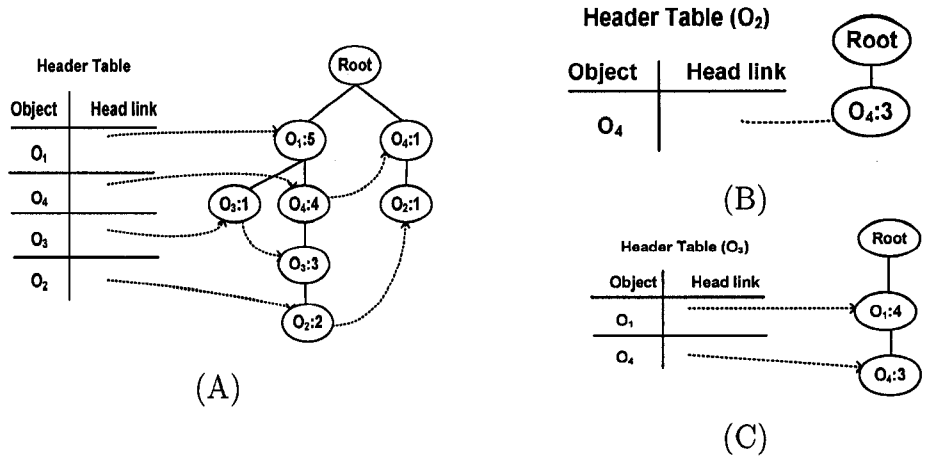


Figure 2.6: FP-Tree Example

nodes in the conditional pattern will have counts equal to the count of the node o_n in that branch. Following extraction of the conditional sub patterns, FP-growth is used to construct the o_n's conditional structure, which is a new FP-tree structure that is built from o_n's conditional sub patterns. The o_n's FP-tree, which has the same structure of the original FP-tree (header table, root, node-link,..etc), represents the projected database of the object o_n; it is built using the same construction procedure used to build the original FP-tree. Next step is the mining process, which proceeds in the same way as before. Starting from the object with the least frequency in the o_n's header table - we will call this object o'_m - the frequent pattern $o_n o'_m$ is generated, and the conditional structure of pattern $o_n o'_m$ is built from the conditional branches of o'_m, which are extracted from o_n's FP-tree. The process continues in the same manner, until no conditional structure can be built. When this happens, the process goes back to considering the next object in the last header table that is more frequent than the current object. The whole process continues recursively until all the objects in the original header table are exhausted. The following steps illustrate the FP-growth algorithm using the database presented in Table 2.2, with minimum support equal to 3.

1. The mining process starts with the least frequent objects in the header table of Figure 2.6-A. That object is o_2, with frequency of 3.

2. The conditional sub patterns of o_2 are extracted by following the node links, starting from the node-link of o_2 in the header table. These patterns include (($o_1 \rightarrow o_4 \rightarrow o_3$):2) and (($o_4$: 1)). The number associated with each branch represent its frequency, which is equal to the count of node o_2 in that branch.

3. o_2's conditional FP-tree structure is built from its conditional sub patterns (Figure 2.6-B). Note that o_2's FP-tree consists of only one node, o_4, which is the only frequent node in o_2's conditional sub patterns.

4. The mining process continues from o_2's conditional structure by considering object o_4, and producing the frequent pattern o_2o_4, with frequency of 3.

5. Since there is no conditional structure for pattern o_2o_4 and the objects in o_2's header table are exhausted, the process goes back to the original FP-tree and considers object o_3. o_3's conditional structure (Figure 2.6-C) is built.

6. The above steps are repeated until all objects in the original header table are passed.

Several algorithms, such as (H-Mine [25], FP-growth* [28], COFI-tree [29], and ITL-tree [26], CT-ITL [27]), have been proposed in the domain of pattern growth. They are variations of FP-Growth; however, FP-Growth is the core algorithm of this approach.

2.3.2 Sequential Patterns

Sequential Patterns is a variation of the Association Rules algorithm. It was first proposed in the domain of transactional databases [32]. Mining sequential patterns is the technique that oversees the generation of subsequences of patterns, from a database of sequences that have frequencies exceeding a given threshold.

Each sequence in the database is in the form $<(p_1, t_1), (p_2, t_2), ...,(p_2, t_n)>$, where p_i, $1 <= i <= n$ is a pattern of objects, and $t_i < t_{i+1}$. t_i is the time of occurrence of

the pattern. For simplicity, we consider only the chronology of pattern occurrences, and ignore the time of occurrence.

Definition 2.3.1. *A sequence* $s' =< p'_1, p'_2, ..., p'_n >$, *is said to be a subsequence of sequence* $s =< p_1, p_2, ..., p_m >$, *if* $n < m$, *and the order of integers* $i_1, i_2, ..., i_n$, *is such that* $p'_1 \subseteq p_{i_1}, p'_2 \subseteq p_{i_2}, ..., p'_n \subseteq p_{i_n}$.

Definition 2.3.2. *Given a database of sequences, the frequency of a sequence, s, is defined by the number of sequences in the database in which s occurs as a subsequence.*

The problem of mining sequential patterns is, essentially, finding sequences that have frequencies exceeding a given threshold [32]. Throughout the literature, sequential pattern algorithms were the natural extension of association rules algorithms [32, 33].

Sequential patterns have been applied to several domains such as DNA sequences [35], intrusion detection [36], web access, and networks alarm management [34, 37].

2.4 Knowledge Discovery in Wireless Sensor Networks

Each sensor node in a wireless sensor network is a potential data source that generates large amounts of data about its surrounding area. However, data generated from sensor nodes is raw data, and must go through stages of interpretation and analysis to extract meaningful information about its environment[40]. Taking a wireless sensor network for fire detection, a query may be posted to the network to request the sensor nodes to report a fire event if the temperature goes above $75^o c$. In this example, the interpretation of the sensor's reading (i.e, $75^o c$) to meaningful information (i.e, a fire event) is simple and straightforward, and can be expressed using a simple query language. However, this case of simple reporting may not always be the case; in many scenarios, there is hidden information about the surrounding environment that requires complex analysis techniques to

extract hidden patterns and correlations from the data. Knowledge Discovery in wireless sensor networks[6, 40] is viewed in that perspective.

Knowledge Discovery is a well known technique in traditional database systems, employed to extracting patterns and useful information from data. The extracted patterns are essential to making decisions about critical issues regarding the domain of data. Knowledge Discovery is a complex process that entails a series of steps including: Domain understanding, knowledge definition, data cleaning, data reduction, and data mining [5, 6]. Refer to Section 2.3 for more details about the Knowledge Discovery process.

Recently, Knowledge Discovery has proven its worth in wireless senor networks, where it is capable of extracting:

1. Knowledge about sensors' data. In this aspect, the Knowledge Discovery process seeks to find hidden patterns and correlations from the sensed data [45, 46, 51]. Data to be used by the knowledge discovery process constitutes raw readings of the sensor nodes.

2. Knowledge about the wireless sensor network. In this category, the Knowledge Discovery process is looking to discover behavioral patterns about the WSN's components in order to improve network performance and its quality of services. [40]. The data to be used in the Knowledge Discovery process for this category is most likely meta-data about nodes' activities.

As in the case of the Knowledge Discovery process in databases[5], Knowledge Discovery in wireless sensor networks has to go through similar steps of domain understanding, knowledge definition, data cleaning, data reduction, and data mining [40]. However, Knowledge Discovery in Wireless sensor networks (KDW) has not been well defined, yet. Furthermore, the transition from KDD to KDW requires careful planning that involves refinement of most of the techniques used in KDD. Moreover, new techniques that account for the limited resources of sensor nodes [1, 40], need to be created for Knowledge Discovery in WSNs.

Two main approaches have been defined for extracting knowledge from wireless sensor network. The first approach aims to accumulate all data at a central server (Sink), before Knowledge Discovery commences [72, 78]. The second approach, on the other hand, applies Knowledge Discovery, or one of its steps, in a distributed manner [74]. The latter approach is preferred because it minimizes the volume of information to be sent to the Sink.

In the next section, we focus on the data mining step of the Knowledge Discovery process, whose purpose is to extract patterns regarding sensor data. These kind of patterns is usually used to gain insight about the phenomena under monitoring. These patterns can be also used to improve the performance of the network. Particulary, we will consider the prediction technique, a data mining techniques that aims to construct a model that predicts future states, based on the current reading. We will also consider the association technique, which aims to discover the temporal correlations between data [2].

2.4.1 Prediction Models in WSNs

Prediction is among the earliest data mining techniques that has been used in wireless sensor networks; it is concerned with the building of a model that predicts future data states of sensor nodes, based on the current data[7]. Two types of predictions can be defined; classification and regression. Classification is used to predict nominal or discrete values, while regression is used to predict continuous or order values [2]. In this section, we provide a comprehensive study of the prediction models proposed in domains of wireless sensor networks, and aspire to answer questions like: what are the types of prediction models that can be extracted from WSNs?; how do WSNs benefit from prediction models?; and what are the methodologies that have been used to build these prediction models? We conclude this section by providing a detailed comparison between the classification and prediction models, to highlight the major similarities and differences between them.

As already known, sensor nodes collaborate with each other to monitor their environment, and report the obtained readings to an access point (Sink). In this setting, prediction techniques are characterized by two aspects that play an important role in reducing the amount of the raw data to be sent to the Sink. First characteristic can be viewed as 'information processing'. As mentioned perviously, most users are interested in meaningful information about the monitored environment, rather than the raw data. For instance, consider a wireless sensor network, consisting of acoustic and seismic sensors, deployed to detect passing vehicles in a specific geographical region [46]. Users may be interested in knowing the type of the passing vehicles, rather than the actual readings of the acoustic and seismic sensors. A prediction model, implemented at sensor nodes, can use classification techniques to identify the type of the vehicle, and send the predicted value (in this case the vehicle type) to the user, instead of sending raw data. Beside reducing the amount of data to be sent to the Sink, this technique 'translates' raw data into meaningful information. This distributed setting makes the prediction process faster than when collecting all data at the Sink, before running the prediction technique. The second characteristic of prediction techniques, that reduces amount of raw data to be sent to the sink is correlation: temporal correlation [60], which involves successive readings of a particular sensor; and spatial correlation [74] (sensors close to each other). Instead of sending sensors' readings every time, a prediction model can be built to foretell future values of each sensor, or the values of other sensors (regression).

Several prediction techniques have been proposed to be integrated with wireless sensor networks. These techniques differ mainly in:

- How to build the prediction model; at the Sink, or in a distributed manner (i.e., at the sensor nodes).
- The type of prediction; regression or classification.
- The technique used to build the prediction model.

In the following two subsections, we will present the two types of prediction; regression and classification, in more details.

2.4.1.1 Regression Based Prediction Schemes

Regression is the type of the prediction that is concerned with predicting the values of continuous variables [2]. Regression techniques have been widely used in wireless sensor networks applications, especially those that require regular reporting of their status [53, 54, 55, 62]. Most of these applications are interfaced with SQL-like query languages; they enable their users to specify continuous queries over the WSN [56, 57, 58]. A common property of these applications, and many other applications, is that users can tolerate a bounded error in the queries' answers [55].

The main goal of Regression-based Prediction algorithm is to build a regression model that predicts future readings of sensors nodes, and exempt the nodes from sending their readings to the Sink. Two main paradigms have been proposed for maintaining regression models:

1. Dual Prediction Scheme [55]. In this paradigm, the same prediction model is built at the Sink node and the sensor node. During the operation time, the sensor node will update the Sink with the new value, if the sensed value has exhibited a significant difference from the predicted value.

2. Single Prediction Scheme [51, 58]. This paradigm builds a single model at the Sink node, or at the sensor node, based on the reading of a subset of sensor nodes.

Following, we will review some of the regression techniques that have been proposed for deployment in wireless sensor networks. However, we will not critique these works, as they differ in the their techniques and applications. We will just provide a brief explananation on how these techniques work.

Olston *et al.*, [54], proposed a dual prediction scheme that uses constant prediction to provide users with an answer that is guaranteed to be within a bounded interval [L,

H]. The bounded interval is computed based on the user's specified precision constraint δ. Let V_t be the current reading of sensor s_i, once the network is deployed, V_t is used to compute the bounded interval of s_i's readings (i.e., $[L_{s_i}, H_{s_i}]$), such that $L_{s_i} = V_t - \delta$. $H_{s_i} = V_t + \delta$. V_t is then transmitted to the Sink, which computes the same bounded interval for sensor s_i. Each sensor s_i is provided with a stream filter. If a future reading (i.e., V_{t+1}) happens to fall within the bounded interval (i.e., $L_{s_i} \leq V_{t+1} \leq H_{s_i}$), it will dropped by the sensors' filter, and the Sink keeps feeding the users with the computed bounded interval (i.e., $[L_{s_i}, H_{s_i}]$). If however, the future reading V_{t+1} does not fall within the bounded interval, it will be sent to the Sink, and a new bounded interval is computed based on the new reading (i.e., V_{t+1}).

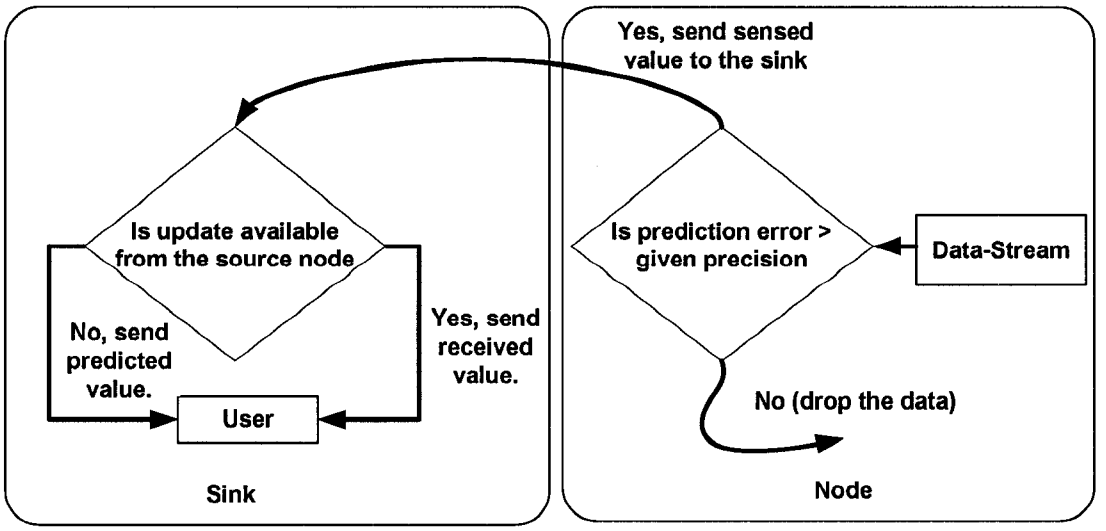


Figure 2.7: Dual Kalman Filtering Model

Dual Kalman Filter (DKF) is a regression-based data gathering framework proposed, by Jan *et al.,* [53]. DKF uses "Kalman filter", a well known linear prediction technique [52], as the main technique to predict future readings of sensor nodes. In this framework, each sensor node, (s_i), and the Sink node run an identical version of the Kalman filter that has been built using the data sensed at sensor node and transmitted

to the Sink. Other parameters describing the environment under monitoring also participate in building the filter. During the monitoring time, sensor s_i transmits the sensed values to the Sink node, only when the prediction error exceeds a given precision value, ϵ_i, on which both nodes have agreed upon before. The prediction error is defined by the difference between the value sensed by the sensor node, and the value predicted by the local Kalman filter. The Sink sends the received value to the user if it was received by the sensor node; otherwise, it sends the value predicted by the Kalman filter. Figure 2.7 shows the basic operations of the Dual Kalman filter (the picture is redrawn from [53]). Although DKF framework provides a good solution for saving communications, and reducing the amount of the data to be sent to the Sink, Santini and Römer [59] noted that "DKF required a prior knowledge or model of the statistical properties of the observed signal". This motivated them to come up with a dual prediction scheme that uses the Least Mean Square (LMS) [61] adaptive algorithm, to build the prediction models at the nodes and the Sink. Adopting LMS has made the scheme more general and therefore, applicable to a variety of real world environments.

PAQ (Probabilistic Adaptable Query system) is a probabilistic-based query answering system, proposed by Tulone *et al.,* [57]. In PAQ system, each sensor samples its readings periodically and maintains a local autoregressive model for the last N readings. Sensor nodes within the same geographical area, in which their readings are correlated, form a cluster. One node in the cluster is distinguished as a cluster leader, that is responsible for building a unified autoregressive model, able to predict the readings of the sensor nodes, from the local models of sensors in its cluster. The leader sends the model to the Sink, that in turn uses it to address users' queries. Each cluster leader is responsible for updating its unified model at the Sink node, once a change is detected in the unified model's parameters. Also, sensors' local models are used, beside being used to build the leaders' model, for outlier detection. Each node compares the sensed value with the value predicted by the local model; if the value exceeds a given precession, the reading is marked as an outlier, and the node relearns its local model.

PREMON (PREdiction-based MONitoring) is a regression-based prediction technique, inspired by the encoding techniques used in video streaming [51]. In this approach, sensor nodes and the Sink use TDMA-based communication, where it is assumed that sensors' readings at each TDMA cycle form the intensity values of pixels in an image. This perspective allows the Sink to use the encoding technique used in $MPEG-2$ to develop a prediction model that predicts the future readings of sensor nodes. Initially, the Sink collects readings from sensor nodes that form a frame of an image. After collecting few frames (i.e., few readings), the Sink instructs the prediction generation unit to build a prediction model. The prediction model is built based on a block matching algorithm used in the MPEG standard. The model is expressed by the motion vectors that were deemed to be valid in a few frames. The prediction model is sent to the sensor node, which uses it to compare the sensed value with the value obtained by this prediction model. An update is sent to the Sink, if there is a significant difference between the sensed value and the predicted value.

The prediction subset is a different approach for regression-based prediction, proposed by Gianluce and Yann-Ale [62]. In this approach, a subset of sensors is identified as being capable of predicting the readings of all sensor nodes in the network. Different prediction subsets are defined to be used, in turn, to distribute the energy consumption between the nodes. Initially, each sensor sends its reading to the Sink node. The Sink first finds a cycle of prediction subsets - constructed through applying a feature selection algorithm, guided by a user-defined accuracy for the prediction errors - and sends an activity schedule, defining the periods within which the sensor may send its readings and the periods when it must remain idle, to each sensor. In each cycle, the reading received from the predicted subset is used to predict the values of the other sensor nodes in the network. For the prediction, the multivariate gaussian models have been used. Although this method has improved sensor nodes' life time, it requires a pre-knowledge of sensors' correlation, to determine which sensors can predict the readings of other sensors.

ASAP (Adaptive Sampling APproach) is a regression-based approach to energy-

efficient and periodic data collection in wireless sensor networks, proposed by Gedik *et al.,* [60]. The main idea behind ASAP is to periodically identify a subset of nodes as sampler nodes, such that all readings of the sampler nodes are sent to the Sink, while readings of non-sampler nodes are predicted using a probabilistic model that is locally built in the network. In this approach, a three-layer network architecture is used for periodic data collection. The first layer is a multi-hop tree consisting of all the nodes in the network, and is rooted at the cluster head. The tree is used to propagate the sensed reading to the Sink. The second layer is a result of partitioning the network into a set of disjoint clusters, in such a way that correlated and neighbor sensor nodes form a cluster. One node in each cluster is distinguished as a cluster head, featuring a tree rooted at the cluster head consisting of all the nodes in the cluster. The tree is constructed to route the data from sensor nodes to the Sink. Clusters formulation and cluster head election occurs periodically to distribute the energy consumption. The third layer consists of further partitioning of each cluster into sub-clusters. Initially, all sensor nodes within a cluster report their readings to the cluster head. The cluster head runs a "correlation-based sampler" to discover the spatial and temporal correlation between sensor nodes, in order to partition the nodes into sub-clusters. One sensor in each sub-cluster is identified as a sampler node. Sampler nodes then report their readings to the Sink, which predicts the reading of non-sampler nodes, through a "model-based prediction algorithm". The prediction models are built based on the parameters received from cluster heads. To keep the prediction model updated, all the sensor nodes within the cluster keep sending their readings to the cluster head, which re-calculates the models' parameters.

Ken is an approximate data collection scheme for wireless sensor networks, proposed by Chu *et al.,* [56]. Ken follows the dual prediction paradigm, in which a pair of identical, synchronized, probabilistic models are built at the sensor nodes and the Sink. A probabilistic model is built for each set of nodes that share the same attributes based on the spatial and temporal correlations. In contrast to the other approaches, Ken updates the models when the model and its mirror become unsynchronized (i.e., the prediction

error exceeds the given parameters), by finding the smallest subset of attributes required to read the model parameters only.

BBQ (Barbie_Q) is a probabilistic-based approach for answering queries proposed by Deshpande *et al.,* [58]. In this approach, users post SQL queries to the Sink, beside the desired information to be retrieved. The query determines the error and the confidence bounds of the answerers. The query processor in the Sink generates a query plan that specifies the features and the sensors to be sampled. It then diffuses this plan in the network. Once the readings are retrieved, they are used to build a probabilistic model, based on time-varying, multivariate Gaussians, that will answer the user's query.

Driven by the need for a complete knowledge about remaining energy in each sensor node, Mini *et al.,* [63, 64] proposed a regression-based prediction approach to construct the network energy map. **Energy map** gives information about the remaining energy in each node in the network, and can be a useful tool for extending the network's life time in different ways; just to mention a few applications [63], (i) helping in making decisions on where to place the Sink, such that it lies in the network part that hosts nodes with the most remaining energy, (ii) choosing the routes that have higher energy nodes compared to other routes, (iii) constructing the query plans for continuous queries. The direct solution to constructing the energy map is to have each sensor node send its remaining energy to the Sink. However, this solution will generate more traffic and as a result, consume more energy. Instead, the authors in [63] proposed to use a regression-based prediction approach to collect the information needed to construct the energy map. In this approach, each sensor builds a local prediction model, based on *"Markov chain"*, to model the energy consumption of the node. Then, each sensor node sends its models' parameters to the Sink, which uses them to construct the energy map of the network. The authors extend their work in [64] to build the energy map under the constraint: that each node will spend a certain amount of energy in building the map.

2.4.1.2 Classification-based Prediction Schemes

In this sub-section, we review some of the works that have been proposed to use classification techniques in wireless sensor networks. Classification is the type of prediction that is used to assign a nominal value to a set of attributes [2]. Like regression techniques, classification techniques have been used to reduce the amount of data to be transmitted to the Sink. In most applications, users are interested in meaningful information about the monitored environment, rather than raw data of sensor nodes. Upon receiving raw data, analysis is performed to it, to extract meaningful information. Classification is among the analysis techniques that can be used to relate raw data, and assign a class label (a useful interpretation) to the set of attribute values received from sensor nodes. For instance, "using ocean overpressure sensors and seismic detectors, to predict a tsunami [45]". Instead of sending all the attribute values of the sensor node, or in the sensor nodes that are close to each other, a class label is assigned to these attributes, based on a classification model that is built locally at the sensor nodes or at the Sink; this class label is sent to the Sink [45, 46]. Beside their efficiency in reducing the amount of data to be sent, classification techniques eliminate one the need for analyzing the data at the Sink.

The research issues that have been considered in the classification techniques for WSNs focus on 'training' the classification model; finding techniques to build the model; deciding where to build the model. In what will follow, we will highlight some of the work that has been proposed to use classification-based prediction in wireless senor networks.

McConnel *et al.,* [45] proposed a general framework for building classification-based predictors in wireless sensor networks. In this framework, each node is responsible for one or more attributes, depending on how many sensors the node has. All sensors are assumed to generate values simultaneously. It is also assumed that all readings in the network, at a given point of time, belong to the global target class in which the Sink is seeking to derive. Instead of collecting all data at the Sink node to predict the class label of the readings, each sensor node builds a local predictor for a pre-defined global

target class and sends its local prediction to the Sink. The Sink decides on the global class target by voting on the received class labels. Each node builds its classification model based on the training data it has received from the Sink. Based on the abilities of the sensor nodes and the type of classification model, training data can be sent to the node, one reading at a time; or all sets of training data are sent at once. Note that each senor node builds a classification model based on its local attributes. Although this framework does not assume a particular classification technique, the authors have used "decision tree" in their simulation [45].

In [46], **Vehicles classifier** for predicting the type of vehicles, based on signals detected by sensor nodes in a distributed wireless sensor networks, is proposed. Acoustic and seismic sensors are deployed in a certain geographical area. Each sensor is equipped with a locale pattern classifier that is able to identify the type of the vehicle, based on a feature vector prepared from the signals sensed by the acoustic and seismic sensors. Once a moving vehicle is detected, the classifier predicts the class of the vehicle and sends the result, along with the probability of the result being correct, to the Sink. The Sink makes a global decision based on the local decisions received from the sensor nodes. The classifier is built at the central node and sent to the sensor nodes. Training is based on data sets that have been collected from real sensor networks. However, there are no assumptions about the methods used to build the classifier. The results are provided using three different algorithms, k-Nearest Neighbor, Maximum Likelihood, and Support Vector Machine [2].

Radivojac *et al.,* [43] studies the problem of **Class Imbalanced Data** in wireless sensor network. As defined in [44], the class imbalanced problem "refers to domains for which one class is represented by a large number of examples, while the other is represented by only a few". The class imbalanced problem exists in wireless sensor networks that are used to detect rare events, such as intrusion detection. In these networks, sensor nodes collect data points at regular time intervals and send them to the Sink. The Sink keeps accepting data points until it builds the first model, using neural network

technique [2]. The model will be able to label the data points into positive examples of interesting events(e.g., rare events), and negative examples. The Sink then sends the model's parameters to sensor nodes. After receiving the model, sensor nodes will send all data points of positive class (rare events), and a subset of the negative labeled data points. In this approach, a reduction on the amount of data is achieved by sending a subset of the negative examples. However, this approach is only limited to a certain type of applications.

In wireless sensor networks, faulty readings, due to faulty nodes, are likely to occur. Krishnamachari *et al.,* [49] proposed a **fault recognition algorithm**, based on Bayesian classifier [2], to predict the correctness of sensor readings. Fault detection is achieved by checking the correlation of senor readings with the readings of neighboring sensor nodes. In this algorithm, once a sensor decides on its reading, let us say v, it will learn how many of its neighbors agreed with its reading (i.e., have reading equal to v), in order to use this value to compute the probability $P(R_i = v|s_i = v, E(v,k))$. This gives the probability that the real reading is equal to v, given that the sensed reading is equal to v, and k neighbors agreed on this value [49]. The sensor node decides that its sensed value is faulty if the Bayesin probability is less than a given threshold.

Table 2.6 shows a comparison between different prediction techniques proposed for wireless sensor networks.

Table 2.6: Wireless Sensor Networks Prediction Techniques

Scheme	Model	Type		Paradigm		Location	
		Regression	*Classification*	*Dual*	*Single*	*Sink*	*Nodes*
Bounded Interval [54]	Constant Prediction	X	-	X	-	X	X
Dual Kalman Filter [53]	Kalman Filter	X	-	X	-	X	X
LMS [61]	Least Mean Square	X	-	X	-	X	X
PAQ [57]	Autoregressive mModel	X	-	X	-	-	X
PREMON [51]	Block-matching	X	-	X	-	X	-
Prediction Subset [62]	Multivariate Gaussian Model	X	-	-	X	X	-
ASAP [60]	Multivariate Gaussian Model	X	-	-	X	-	X
Ken [56]	Markovian Models	X	-	X	-	X	X
BBQ [58]	Multivariate Gaussian Model	X	-	-	X	X	-
Energy Map [63]	Markov Chains	X	-	-	X	-	X
Distributed Prediction [45]	Decision Tree	-	X	-	X	-	X
Vehicle Classification [46]	K-Nearest Neighbor	-	X	-	X	X	-
Class-Imbalanced [43]	Neural Networks	-	X	-	X	X	-
Fault-recognition [49]	Bayesian	-	X	-	X	-	X

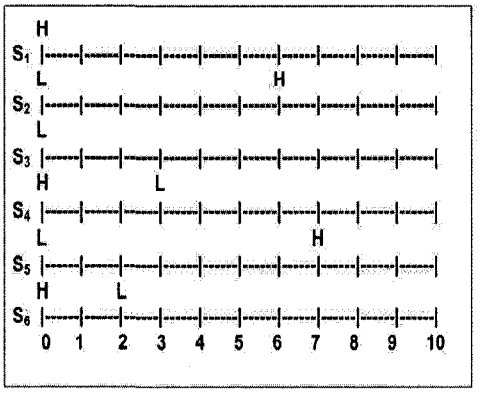
2.4.2 Association Rules for WSN

Association rules is the data mining technique that is concerned with discovering the correlations between the objects that lie within the same context. Association rules have been used in different domains, such as business analysis [11] and medical applications [13]. As in the case of prediction, Association Rules have been used to generate patterns pertaining to wireless sensor networks and their underlying domain. In this section, we review some of the attempts that have been proposed to use Association Rules with sensor networks, and highlight the main techniques that have been used to generate these rules.

2.4.2.1 Mining Inter-Stream Associations

Loo at al., [72] studies the problem of mining associations between sensors' values in data streams generated from sensor nodes, in a particular wireless sensor network. Their technique depends on a data model that stores data reported from sensor nodes, and presenting it in a way that facilitates the adaption of loosy counting algorithm [73] that makes one pass analysis of the data. In this data model, sensors are assumed to take values from a finite discrete number of values. A quantization method is applied for the continuous values. The time is divided into equal-sized intervals, and snapshots from the sensors' readings are taken whenever there is change to these readings. These snapshots are stored in a database in the form of contexts. Taking snapshots at states' changes reduce the redundancy in the database. To overcome the problem of random state changes in sensor nodes, each context is associated with a weight value indicating the number of intervals for which this context is valid (i.e., for how long the readings in the context are kept unchanged). To illustrate this model, we use the same example provided in [72]. Figure 2.8-A (Figures A and B are redrawn from [72]) depicts the states of six sensor nodes during a period of 11 seconds. Each sensor takes a value from two possible states (High H, Low L). For instance, sensor s_2 exhibits state L, at time 0; and state H, at time 6. The first context is {(s_1, H), (s_2, L), (s_3, L), (s_4, H), (s_5, L), (s_6,

H)}, which is valid for two seconds before a state change occurs. Figure 2.8-B shows the extracted database.



(A) Sensors' reading

Context	S_1	S_2	S_3	S_4	S_5	S_6	wight
1	H	L	L	H	L	H	2
2	H	L	L	H	L	L	1
3	H	L	L	L	L	L	3
4	H	H	L	L	L	L	1
5	H	H	L	L	H	L	3

(B) The database

Figure 2.8: Inter-stream Mining Example

The proposed data model maps the problem of mining associations between sensors' values to the traditional association mining problem. Each possible sensor state is considered as an object; and a pattern is a set of sensors' states. For instance (s_1 = L, s_4 = H) is one of the possible patterns. The support of the pattern is defined by the total length of non-overlapping intervals, in which the pattern is valid. To facilitate support counting, sensors' data is represented by interval lists. The interval list is a list of pairs containing the start time and end time for which the patterns are valid. For instance, the interval list of the pattern s_2 = L is (i.e., IL(s_2 =L)) is [(0-6)].

To generate the frequent patterns, the authors [72] adapted the loosy counting algorithm to make it applicable for their data model. The advantage of the loosy counting algorithm is that it can make online analysis with one pass of the database [73]. Lossy counting does not generate the exact frequent patterns (i.e., value sets), however, it provides a solution with a bounded error.

2.4.2.2 Mining Spatial Temporal Event Patterns

Mining spatial temporal event patterns is another attempt to link the problem of mining sensor data to the association mining problem, proposed by Kay Römer [74]. Römer's

approach took into consideration the distributed nature of wireless sensor networks, and proposed an in-network data mining technique, to discover the frequent patterns of events with certain spatial and temporal properties. In this approach, each sensor node is aware of the events that are within a certain distance from itself (this distance may be a Euclidean distance or number of hops). The sensor then collects these events and applies a mining algorithm to discover the pattern that satisfies a given parameters.

Römer's mining parameters approach includes minimum support S, minimum confidence C, maximum scope, and maximum history. Every node in the network collects the events from the neighbors within the maximum scope and keeps a history of their events for a duration of the maximum history. After that, each node applies a mining algorithm to discover the frequent patterns of the form:

$$A_1 \wedge A_2 \wedge \cdots A_m \Rightarrow E[S, C].$$

which means that if all the predicates in the rule antecedent become true, then event E may occur at the node with support S and confidence C. Each predicate in the rule antecedent is in the form $A_i = (E_i, D_i, T_i, N_i)$. A_i is true if and only if "eventE_i occurred N_i times at a distance D_i from the node and T_i time units before the the occurrence of event E" [74].

To map Römer's framework to the traditional association mining problem, a quantization technique is used to quantize the continuous parameters like distance, time offset, and number of occurrence of the events. For instance, the distance parameter can be divided into two variables, near (between 0 and 5 meters), and far (between 5 and maximum scope). Nodes start collecting events from their neighbors at regular intervals, each interval is called an epoch. Each node maintains a table of $\#E \times \#D$ (i.e., number of events times number of distance's partitions) columns. A cell corresponding to the column (e_i, d_i) is incremented, once an event e_i is received from a node within the distance d_i. The table contains one row for each possible epoch in the given maximum

history. At the end of the historical period, the table is grouped and summed based on the time partitions. A context is then created for each epoch; that consists of the the events occurred at the node during the epoch's time, in addition to each possible event e_i occurring n times in the neighboring nodes at distance d and time offset t (i.e. event e_i is included in the epoch if the cell $(e_i, d_i, t_i) >= n$).

Due to the large number of possible frequent patterns, Römer suggested to generate the closed or the maximal patterns, which are much smaller than the set of frequent patterns. A pattern P is maximal if it is not a subset of another frequent pattern [77], and closed if non of its proper supersets has the same support [76].

2.4.2.3 Window Association Rule Mining

Missing values is a frequent problem in wireless sensor networks, due to the low battery of sensor nodes, transmission errors, and sensor failures [9, 38]. Different techniques have been used to estimate the value of missed reading, like expectation maximization algorithm [79]. In this section, we focus on one method that has used Association Rules for estimating sensors' missing values.

Halatchev *et al.*, [78], proposed an association rule mining framework to tolerate the missed readings that resulted from the loss and corruption of messages while they being routed from sensor nodes to the Sink. Sensor readings are stream in nature, so applying an association mining algorithm like Apriori directly to the stream of data is not possible. This situation led authors to propose the Data Stream Association Rule Mining (DSARM) framework that transforms the Apriori algorithm [12] to make it applicable to the data stream received from sensor nodes.

Several modifications that have been made for the Apriori algorithm to render it usable for sensors stream. First, rules are generated between pairs of sensor nodes, instead of generating all the possible rules. Second, association between a pair of sensors is evaluated with regards to a particular state of the sensor nodes. This modification lead to generate rules of the form $(s_1 \Rightarrow s_2/st)$, meaning that s_1 determine s_2 with regard

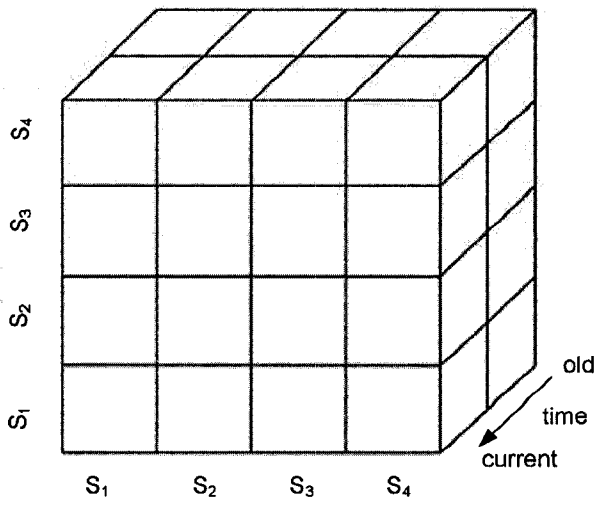


Figure 2.9: The Cube Model

to state *st*. Finally, sliding window technique has been used to generate the association between sensors within the given window size.

To allow for fast estimation for the missed reading, the authors proposed a cube data structure to efficiently store sensor readings. Sensors' data arrive in rounds. The cube is used to keep track of the state of individual sensors and the pairs of sensors that have the same state (i.e., reading). The cube's horizontal and vertical dimensions are the sensors' identifers, and the depth dimension is the number of rounds(i.e. window size). Cell(i,j,r)=*st* in the cube means that sensors s_i and s_j have the same state *st* at round r. Figure 2.9 shows the proposed data cube(Figure redrawn from [78]).

Given a round r and a missed reading from sensor s_m, the estimation process starts by determining all the frequent states of sensor s_m (i.e. those states that have support greater than, or equal to, a given minimum support). Then, for each frequent state *st*, all the possible rules of the form $(s_i \Rightarrow s_m/st)$ that meet the given minimum support and confidence, are generated. After that, the weight of the contribution of each sensor, appearing in the antecedent of the frequent rules toward the missed reading with regard

to state st, is computed. This weight is based on the number of state machete between sensor s_i and sensor s_m, within the given window size. The missed value is then estimated, based on the weight of each sensor.

2.5 Summary

Throughout this chapter, we have reviewed the main components pertaining to this thesis. We started by giving an overview of wireless sensor networks, their applications, their characteristics, and their routing techniques. Then, we presented the Knowledge Discovery process and briefly described its steps with more focus on the data mining step and its techniques; especially, the Association Rules technique, which influences most of the patterns proposed in this thesis. Finally, we introduced the Knowledge Discovery process from the wireless sensor networks' perspective, and highlighted the main challenges regarding this process in sensor networks. A comprehensive survey for most of the works that have been proposed to use Knowledge Discovery techniques in wireless sensor network have been provided.

Chapter 3

Sensor Association Rules

3.1 Introduction

In this chapter, we present the Sensor Association Rules, the first kind of behavioral patterns that is proposed in this thesis. Sensor Association Rules is an attempt to capture the temporal correlation between sensor nodes in a particular Wireless Sensor Network (WSN), in order to define the set of sensors that detect events in common time intervals. Sensor Association Rules plays an important role in enhancing the performance of WSNs, and thus improves the Quality of Services (QoSs) of WSNs. Generation of sensor rules from WSN goes through a series of planned steps in the Knowledge Discovery Process; these steps include: defining the type of knowledge to be extracted (i.e., a formal definition for the sensor rules), preparing the data needed for generating the association rules, and implementing a data mining step that 'manufactures' these rules. The Knowledge Discovery Process in WSNs, opposed to in databases, entails the alleviation of constraints that are inherent in WSNs. An example of resolving problems that characterize WSNs include minimizing communications necessary for extracting the data (required by the Mining Step), [1]-sensor nodes possess limited energy; and devising a fast mining algorithm that meets the requirements of the critical class applications.

We will consider the first two steps in the Knowledge Discover Process that pertain

to generate Sensor Association Rules: The Knowledge Definition Step, which defines the Sensor Association Rules algorithm; and the Data Preparation Step, which features the extraction mechanisms required to collect data (from sensor nodes). Three different data extraction mechanisms are proposed for preparing the data needed in the Mining Step. These mechanisms can be as simple as ad hoc solutions, or as sophisticated as reducing the amount of data by eliminating its redundancy, to a certain extent. It is important to recall that the data in our context refers to data that is needed in the mining process, and differs from the actual (raw) readings of the sensor nodes.

This chapter is organized as follows: Section 3.2 provides a formal definition for Sensor Association Rules. Section 3.3 presents the three extraction mechanisms that are used for preparing the data needed for generating sensor association rules-the data mining step for generating sensor rules will be discussed in the next chapter-Section 3.4 presents a performance evaluation for the proposed data extraction mechanisms. Section 5.4 summarizes the chapter.

3.2 Sensor Association Rules: Formal Definition

Our definition of Sensor Association Rules is based on the definition of Association Rules that was proposed in the domain of transactional databases [12]. The latter definition offers little on amplifying on the subject of Association Rules for wireless sensor networks, where the sensors themselves happen to be the main objects in the extracted rules. It is important to note that data referred to in this section is behavioral data that describes sensor activities in the WSNs, not the values of sensor events.

Let S = $\{s_1, s_2, ..., s_m\}$ be a set of sensors in a particular sensor network. We assume that the time is divided into equal-sized slots $\{t_1, t_2, ..., t_n\}$, such that t_{i+1} - t_i =λ for all $1 < i < n$, where λ is the size of each time slot; and $T\_his = t_n - t_1$ represents the historical period of the behavioral data defined during the data extraction process. We also refer to $P = \{s_1, s_2, ..., s_k\} \subseteq S$ as a pattern of sensors.

Definition 3.2.1. *A sensor database, (DS), of behavioral data is defined to be a set of epochs in which each epoch is a couple $E(T_s, P)$, where P is a pattern of sensors that report events within the same time slot. T_s is the epoch's time slot.*

Throughout this thesis we may refer to the database (DS) as behavioral data or metadata that contain encodings of periods of sensor activities during their operations. As mentioned above, this data differs from raw sensor readings. Table 3.1 shows an example of a behavioral database that consists of four epochs. The first tuple in the database encodes the fact that each senor (s_1, s_2, and s_3) detected at least one event within the time slot number 1. For simplicity, we refer to the time slot by an integer number rather than the actual time.

Table 3.1: Behavioral Database (DS) Example

ts	***P***
1	$s_1 s_2 s_3$
2	$s_1 s_2 s_3$
3	$s_1 s_4$
4	$s_1 s_2 s_3 s_4$

Definition 3.2.2. *Let P_1 be a pattern of sensor nodes, such that $P_1 \subseteq S$. We say that an epoch $E(T_s, P)$ supports P_1 if $P_1 \subseteq P$.*

Definition 3.2.3. *The support of pattern P_1 in DS is defined as the number of epochs in DS in which P_1 is a subset of their patterns:*

$$\textbf{\textit{Support}}(P_1,\ DS) = \mid \{E(T_s, P) | P_1 \subseteq P, E \in DS\} \mid.$$

Sometimes we refer to the value **Support** (P_1, DS) by frequency of P_1 or support count of P_1.

Definition 3.2.4. *A pattern P is said to be a frequent sensor association pattern if its support count is greater than or equal to a given minimum support.*

Definition 3.2.5. *Let P be a frequent association pattern. Sensor Association Rule is defined by the implication* $P' \Rightarrow P''$, *where* $P' \subset S$, $P' \cap P'' = \phi$, $P'' \subset S$, *and* $P' \cup P'' = P$.

Definition 3.2.6. *The support of the rule* $(P' \Rightarrow P'')$ *is the support of pattern* $(P' \cup P'')$ *in DS, while the confidence of the rule is defined by:*

$$\textbf{\textit{Conf}}(P' \Rightarrow P'') = \textbf{\textit{Support}}(P' \cup P'', DS) \,/\, \textbf{\textit{Support}}(P', DS).$$

We say that a rule is of interest to the targeted application if its support and confidence are greater than or equal to a given minimum support (*min_sup*) and minimum confidence percentage (*min_conf*). *Min_sup* represents the minimum number of epochs that the support of the rule should satisfy.

Given a database of epochs generated at a particular time slot size and historical period, and minimum support and minimum confidence; mining Sensor Association Rules involves generating all the interest rules presented in the behavioral data. Mining association rules can be broken into two steps [11]:

1. Generating the frequent patterns (i.e., those that have support >= *min_sup*).
2. Generating the rules that satisfy the *min_conf* restriction.

Generating Sensor Association Rules can be a straightforward process that does not expend long running time, provided that the frequent sensor association patterns are defined. In this thesis, we will focus on:

1. How to extract the data required for the mining process (i.e., how to prepare the DS) efficiently.

2. How to generate the patterns that meet a given minimum support (i.e., the frequent association patterns) efficiently.

Now that the Sensor Association Rules have been defined, the next section presents the data extraction mechanisms that prepare the meta-data (i.e., the DS), which is needed in the mining process.

3.3 Data Extraction Methodologies

In order to prepare the data needed for generating Sensor Association Rules, each sensor node should monitor its activity over time, and inform the Sink about the time slots in which events were detected. We will refer to the information sent to the Sink by sensor behavioral data. In this section, we will present three possible methodologies for extracting the sensor behavioral data from wireless sensor networks. The proposed mechanisms are: (i) Direct Reporting; in this mechanism, sensor behavioral data is transferred to the Sink without any processing by the sensor node. (ii) Distributed Extraction; in this mechanism, the limited resources of the sensor nodes are employed to minimize the number of the messages needed to encapsulate the behavioral data, by disregarding unnecessary data. (iii) In-Network Reduction; in this mechanism, the number of messages needed to encapsulate the data is minimized by decreasing redundancy presented in sensor behavioral data. In what will follow, we will describe these methodologies in more details

3.3.1 Direct Reporting

Direct Reporting is an ad-hoc solution to extract behavioral data from sensor nodes. This mechanism makes no assumptions about the capabilities of sensor nodes. Each senor reports its behavioral data to the Sink, even if it is not participating in formulating the association rules.

The extraction process in Direct Reporting starts with the application that provides the mining parameters to the Sink; these parameters include the time slot size (λ), historical period (T_his) and minimum support (min_sup). The Sink then broadcasts the time slot size and the historical period to the nodes in the network. Each node keeps track of the time: At the end of a time slot, the node checks for detected events, and if there are any, it sends a notification message that contains its identifier and the time slot number where the event occurred (an integer number used to refer to the current time slot) to the Sink-the notification message should not be confused with the actual message that carries the sensed value-The time is monitored, and at the end of each time slot (with additional delay), the Sink checks the received message and creates an epoch for the current time slot consisting of the time slot number and a pattern of sensor identifiers (extracted from the received messages that carried the same time slot number). The Sink then stores this epoch in the database.

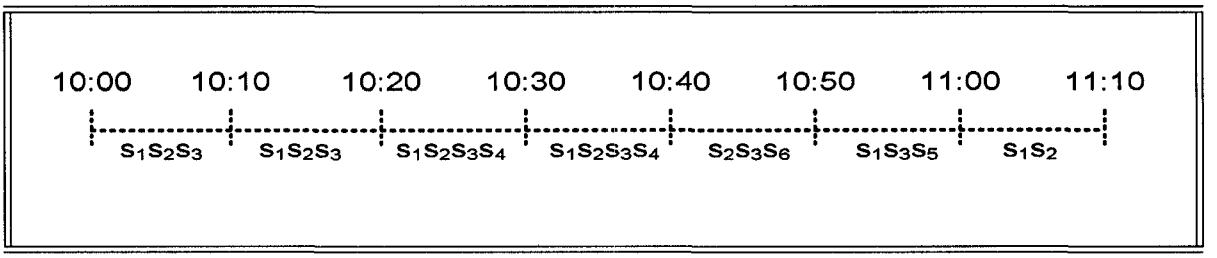


Figure 3.1: Detected Events for a Historical Period of 70 minutes.

Let us consider the following scenario as an illustrative example for Direct Reporting. Let S = $\{s_1, s_2, ..., s_6\}$ be the set of sensors in a particular sensor network. Assume that the time slot size is equal to 10 minutes; the historical period to 70 minutes, and the starting time to 10:00. Figure 3.1 shows the sensors that have detected events in the WSN for the historical period of 70 minutes. At the end of the first time slot (10:10), sensors $\{s_1, s_2, s_3\}$ send the following messages respectively: $M_1(1, s_1)$, $M_2(1, s_2)$, and $M_3(1, s_3)$, which contain the epoch time slot number where an event was detected, and the sensor identifier. An absolute integer is used to refer to a particular time slot instead of using actual time. At time (10:10 + α), the Sink formulates the first epoch $E(1, (s_1 s_2 s_3))$ and

stores it in the database. The same process is repeated periodically at the end of each time slot until the end of the historical period. Table 3.2 shows the extracted epochs after a historical period of 70 minutes. Algorithm 1 presents a formal description for the Direct Reporting.

Table 3.2: The Behavioral Database Using Direct Reporting (DS)

Ts	***Epoch***
1	$s_1 s_2 s_3$
2	$s_1 s_2 s_3$
3	$s_1 s_2 s_3 s_4$
4	$s_1 s_2 s_3 s_4$
5	$s_2 s_3 s_6$
6	$s_1 s_3 s_5$
7	$s_1 s_2$

3.3.2 The Distributed Extraction

Distributed Extraction is designed to put more computational load on the sensor nodes by equipping each sensor with additional storage space to store behavioral data in the historical period.

The extraction process starts when the Sink diffuses the mining parameters to all nodes in the network. These parameters include: the minimum support, the time slot size and the historical period. Upon receiving the mining parameters, each sensor establishes a local buffer, B, of size (T_his/λ), one for each time slot. We will refer to the buffer entry of slot t_i by $B(t_i)$. Initially, all bit entries in the buffer are unset. After that, sensors keep track of the time, and at the end of each time slot each sensor checks for any detected event within this time slot. If there is such an event, the bit entry corresponding to this time slot is set. At the end of the historical period, each sensor examines its local buffer: If the number of set bits is greater than or equal to the given minimum support, the node will establish a message (or a series of messages depending on the packet size) containing the sensor identifier and the time slot numbers in which the corresponding bits are set. Then, the sensor sends this message to the Sink. The Sink waits until it

Algorithm 1 Direct Reporting

```
Sink:
Broadcast mining parameters (T_his,λ);
Slot_Number = 1;
Time = current_time;
While (current_time <= Time + T_his)
   If ( current_time <= Time + (Slot_Number * λ))
      Do nothing;
   Else
      P = {The set of sensors' identifiers in the received messages};
      E = (Slot_Number, P);
      Insert(E, DS);
      Slot_Number = Slot_Number +1;
   End{if}
End{while}

Node:
Upon receiving the mining parameters
   Slot_Number =1;
   Time = current_time;
   Reported =False;
   While (current_time ≤ Time + T_his)
      If (current_time ≤ Time + (Slot_Number * λ))
        If ((there is a detected event) and (not Reported))
           M=(Slot_Number, Sensor_ID);
           send M to the sink;
           Reported = True;
        End{if}
      Else
        Slot_Number = Slot_Number + 1;
        Reported = False;
      End{if}
   End{while}
```

receives all possible messages from the sensor nodes before it restructures the data in the messages in such a way that places all sensor nodes that reported an event at the same time slot in the same epoch. This epoch is then stored in the database.

As an example, let us reconsider the events depicted in Figure 3.1. Physically, each sensor will maintain a buffer of length seven, one entry for each time slot. A buffer entry is set if there is a detected event in that time slot. Figure 3.2 shows the activity buffers for the sensor nodes in our example. Assume that the minimum support is 2, then at time 11:10, the end of the historical period, sensors s_1, s_2, s_3, and s_4 will formulate the following messages: (s_1, [1, 2, 3, 4, 6, 7]), (s_2 ,[1, 2, 3, 4, 5, 7]), (s_3 , [1, 2, 3, 4, 5, 6]), and (s_4 , [3, 4]) respectively. These messages are then sent to the Sink. Note that there are no messages to send from nodes s_5 and s_6 since the number of set entries is less than the

required minimum support. The Sink waits until it receives all messages and formulates the epochs to be stored in the database. Table 3.3 shows the extracted database using the distributed extraction. Algorithm 2 presents a formal description for the Distributed Extraction mechanism.

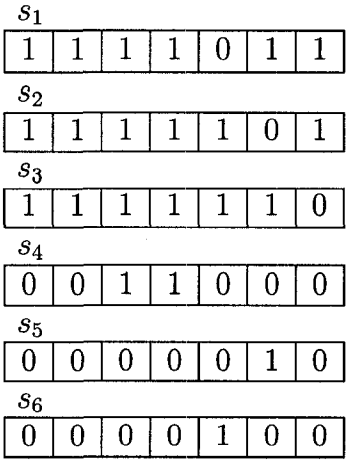


Figure 3.2: Sensor Buffers Example

Table 3.3: The Behavioral Database Using Distributed Extraction (DS)

Ts	***Epoch***
1	$s_1s_2s_3$
2	$s_1s_2s_3$
3	$s_1s_2s_3s_4$
4	$s_1s_2s_3s_4$
5	s_2s_3
6	s_1s_3
7	s_1s_2

Algorithm 2 Distributed Extraction

Sink:
Broadcast mining parameters (T_his, λ, min_sup);
Upon receiving all messages
  For Slot_Number = 1 to (T_his / λ)
    P = The set of the sensors' identifiers that share the same time slot
    E=(Slot_Number, P);
    Insert(E, DS);
  End{for}

Node:
Upon receiving mining parameters
  Slot_Number =1;
  Time = current_time;
  While (current_time $\leq$ Time + T_{his})
    If (current_time $\leq$ Time + (Slot_Number * λ))
      If there is a detected event
        Set buffer[slot_Number];
      End{if}
     Else
      Slot_Number = Slot_Number + 1;
    End{if}
  End{while}
  If (number of set bits $\geq$ min_sup)
    AS = {The set of time slots that have set bits};
    M = (Sensor id, AS);
    Send M to the Sink;
  End{if}

3.3.3 In-Network Data Reduction Mechanism

Although the Distributed Extraction mechanism achieves good reduction in number of messages that need to be sent to the Sink, the process of collecting sensor behavioral data is still a costly process due to the limited resources of sensor nodes. From the structure of behavioral data (i.e., the messages sent by sensor nodes at the end of the historical period), we can see that there is high redundancy among the different sensor data behavior; this redundancy can be removed thus reducing the size and number of messages needed to report behavioral data (i.e., several sensor nodes share the same time slot numbers in which their events were detected). Traditional data aggregation techniques (e.g., sum, average, count, etc...) are unsuitable for use with sensor data behavior. It is important to design a technique that will reduce the volume of behavioral data. In this section, we propose a data gathering algorithm that uses an In-network Reduction technique to reduce the amount of behavioral data that need to be sent to the Sink.

The proposed in-network data reduction is implemented on top of a data gathering tree, we refer to it as Minimum Nodes Data Gathering Tree (MNDGT), and in cooperation with the distributed extraction mechanism discussed in the pervious section. The MNDGT takes in the consideration that not all sensor nodes are going to participate in formulating the sensor association rules and it is constructed in such a way to include just those nodes that will participate in formulating sensor association rules plus the nodes that are needed to maintain the minimum distance to the Sink.

3.3.3.1 Preliminary

In this section, we provide the key definitions pertaining to the construction of the Minimum Nodes Data Gathering Tree (MNDGT), which is the tree structure used to route the sensor behavioral data to the Sink.

The network architecture is assumed to consist of a set of n sensor nodes, $S = \{s_1, s_2, ..., s_n\}$. Time is divided into equal sized slots, of size λ each (among the mining

parameters provided by the user). The In-network Reduction mechanism follows the same strategy in Distributed Extraction to profile sensor behavior. Each sensor has a buffer B of size (T_his/λ)-one entry for each time slot-Sensors control their activities by keeping track of the time, and at the end of the current time slot, each sensor sets the buffer entry corresponding to its slot if an event has been detected during this time.

Definition 3.3.1. *Let s_i be a sensor node in the set, S. $AS(s_i) = \{t_1, t_2, ..., t_m\}$, such that $m <= (T\_his/\lambda)$ and $B(t_j) = 1$, for all $1 <= j <= m$, is then defined as the Activity Set of sensor s_i.*

For instance, in Figure 3.2, $AS(s_2) = \{1, 2, 3, 4, 5, 6\}$ (Recall that we refer to the time slot by an integer number rather than the actual time).

Definition 3.3.2. *s_i, $1 <= i <= n$, is defined as an active node if $|AS(s_i)| >= min\_sup$, otherwise it deemed inactive.*

Definition 3.3.3. *$NL(s_i)$ is the set of all neighbors of node s_i that are within its radio transmission.*

Definition 3.3.4. *Candidate parents of node s_i, $CP(s_i)$, is the set of nodes in the set $NL(s_i)$ that are at the minimum distance, in number of hops, from the Sink.*

The goal of the Data Gathering mechanism is to route the Activity Sets of the active nodes to the Sink. The next section will illustrate the process of building the MNDGT that will be primal in routing the Activity Sets to the Sink.

3.3.3.2 MNDGT Construction Process

At the end of the historical period, the behavioral data collected by the sensor nodes should be delivered to the Sink. Any routing mechanism can be used for this purpose;

however, it is preferable to have a special mechanism for delivering the behavioral data, in order to separate it from the actual event delivering mechanism that is used by the network. In this section, we present the construction process of the Minimum Nodes Data Gathering Tree (MNDGT), which will be used to deliver the Activity Sets of the sensor nodes to the Sink. MNDGT 'envisions' a tree structure, and designs it in such a way that the Activity Sets of different nodes can meet at intermediate nodes where a reduction technique can take place.

The MNDGT construction process starts at the end of the historical period. We assume that each node has prior knowledge about the mining parameters (*min_sup*, λ, and *T_his*), and that it has already prepared its Activity Set. The Sink begins with broadcasting a construction message that consists of three fields: sender identifier, hop count, and support count. The hop count indicates the distance from the sender to the Sink. Support count is the cardinality of the sender's Activity Set. Each node 'in range' of the Sink will receive the construction message; update its candidate parents' set; and rebroadcast the construction message to its neighbors after it modifies the sender field, the hop count, and the support count fields. The candidate parents of node x is the set of nodes that are the closer to the Sink than node x, and which can be used as gateways. Initially, the candidate parents' set of the node is empty, but once a message from a node that is closer to the Sink than the node in question is received, the sending node is added to the candidate parents' set, along with its support count. Later, if the node receives a construction message from another node that happens to be closer to the Sink than any of the other nodes in the candidate set, the 'closest' node is added to the candidate parents' set; the old candidates are removed; and a new construction message is broadcasted. It may be that more than one node in the candidate parents' set are at an equal distance (same hop count) from the Sink.

Figure 3.3 shows an example of a WSN's state after diffusing the construction message. In this example, the network consists of 16 nodes deployed in a grid. Each node communicates with a maximum of 8 neighbor nodes (left, right, top, down, and diagonal

nodes). Nodes s_3 and s_4 are in the range of the Sink node. Labels in a sensor node refer to the sensor identifier, its hop count, and its support count (i.e., the cardinality of node Activity Set). For instance, the top left node in Figure 3.3 (s_2:2:3) refers to node s_2, which is 2 hop counts from the Sink, and has a support count equal to 3. CP is the set of the node's candidate parents; for instance, the candidate parents of node s_{10} are s_5 , s_6 , and s_7 . The Gray color denotes inactive nodes.

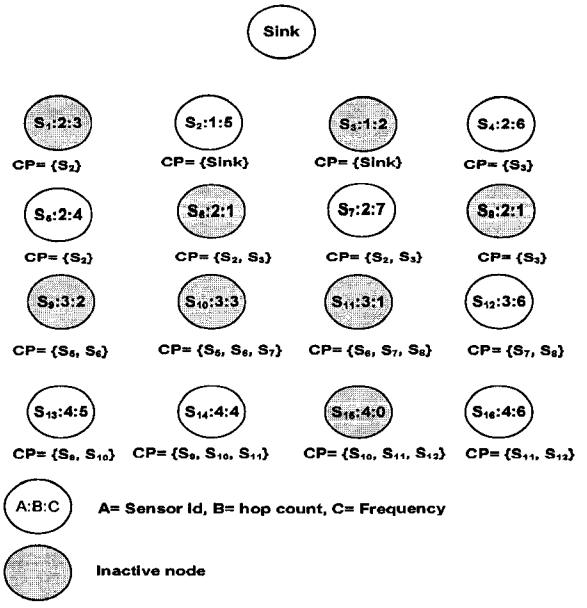


Figure 3.3: MNDGT Example - Construction Phase

The next step in the MNDGT construction process is determining the links between the nodes (i.e., the actual parents). Each active node scans its candidate parents' set and chooses the node with the highest support count, to be its parent. Then, the parent is informed of the node's decision by sending a build message that carries the node's identifier. Another kind of nodes that sends build messages are called "mandatory" nodes. A node is mandatory in the MNDGT if it is inactive and has received a build message. The mandatory nodes are necessary to guarantee that each node can reach the Sink with the minimum number of hops. Each parent node maintains a list of its

children for further communication. Figure 3.4 shows the MNDGT for the network in Figure 3.3. Nodes s_3 and s_{10} are mandatory nodes: s_3 received a build message from s_4, and s_{10} received a build messages from nodes s_{13} and s_{14}. Figures 3.5 shows the format of the construction and build messages. Algorithm 3 shows a formal description for the MNDGT's construction process.

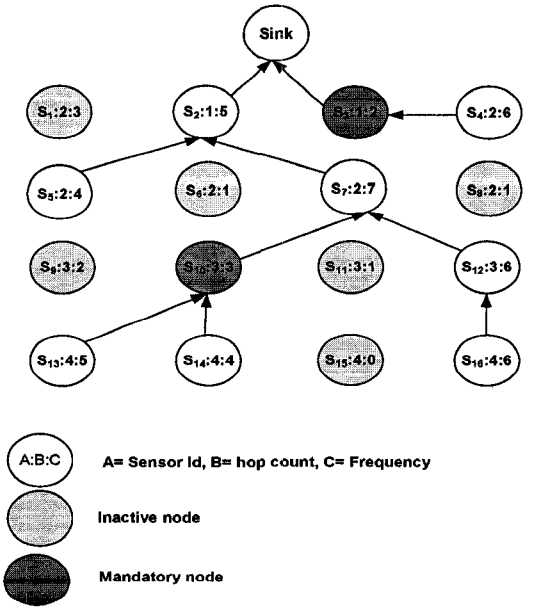


Figure 3.4: MNDGT Example - Build Phase

Construction message

msg_type	*sender_id*	*hop_count*	*support*
C			

Build message

msg_type	*child_id*
B	

Figure 3.5: Construction and Build Messages Format

Algorithm 3 MNDGT's Construction Process

```
% hop_count is initialized to ∞ for all nodes
Construction message propagation phase
  Sink
    Sink.hop_count = 0 ;
    Sink.support = ∞ ;
    C_msg.sender_id=Sink;
    C_msg.hop_support =Sink.hop_count;
    C_msg.support =Sink.support;
    C_msg.min_sup =min_sup;
    Send(C_msg, NL)
  Node_i
    Upon receiving C_msg
      If (C_msg.hop_count ≤ hop_count(node_i))
        node_i.hop_count = C_msg.hop_count +1;
        Update (CP(node_i));
        C_msg.sender_id =node_i;
        C_msg.hop_count =node_i.hop_count;
        C_msg.support =node_i.support;
        Send(C_msg, NL(node_i))
Build message propagation phase
  For each s_i ∈ S
    If((s_i.support) ≥ min_sup) or (Build message received)
      Let p be the parent in CP(s_i) with maximum support count;
      B_msg.child_id=s_i;
      Send(B_msg, p));
```

3.3.3.3 In-Network Data Reduction

The MNDGT presented in the last section is used to route the Activity Sets from the sensor nodes to the Sink. However, without any reduction mechanism, these nodes serve as mere routers for the messages. In this section, we present an in-network reduction mechanism to be integrated with MNDGT in order to remove some of the redundancy in the Activity Sets while they are routed to the Sink. Recall that a sensor Activity Set consists of the time slot number where the sensor detected events.

Sensors data activities differ from other data in the following aspects:

1. Data aggregation functions (average, summation, etc...) cannot be applied to the Activity Sets.

2. The activity data is represented using sets, which limits the operations that can be applied to the data, and makes it difficult to find a function-and its inverse-that is able to compress and decompress data without losing information.

From the above, it can be concluded that it is not feasible to apply the standard aggregation functions to reduce the amount of the data. However, there is a possible way that can be used to eliminate the redundancy between the Activity Sets. If we can find a set that is common among a number of Activity Sets, we will refer to this set as a Reference Set. The difference operation is applied to the Activity Set and the Reference Set, and the results of the difference operation are sent to the Sink: The number and size of messages are reduced. In order to increase the possibility of reduction, the sensors are partitioned into different groups, with each group sharing one Reference Set. One possible way of partitioning the sensor nodes is to use the structure of MNDGT; by having a node and its children share the same Reference Set, and each node in the group participating in formulating the Reference Set. We will start defining participation, and then illustrate the reduction technique in more detail using a running example.

The definition of participation and Reference Sets is recursive, so we have to take into account the concept of Reference Sets.

Definition 3.3.5. *Let s_i be the parent node of node s_j in the MNDGT; let AS and RS be the Activity and Reference Sets, respectively,*

$$PS(s_i) = AS(s_i) \qquad s_i \, active \, node.$$

$$PS(s_j) = \left\{ \begin{array}{ll} AS(s_j) & s_j \, active \, leaf \, node \\ RS(s_j) & \end{array} \right\}$$

is defined to be the Participating Sets of sensors s_i and s_j.

Definition 3.3.6. *Let s_i be a parent node in the MNDGT; let $C(s_i)$ be the set of its children:*

$$RS(s_i) = \left\{ \begin{array}{l} \left(\bigcap_{\forall s_j \in C(s_i)} PS(s_j) \right) \cap PS(s_i) \quad s_i \, active \, node \\ \\ \bigcap_{\forall s_j \in C(s_i)} PS(s_j) \quad s_i \, mandatory \, node \end{array} \right\}$$

is defined to be the Reference Set of the sensor s_i.

The Reference Set of a parent node is computed by intersecting all of the Participation Sets of the children nodes and the Participation Set of the node itself, if it is active. The main computation will occur at the parent node. The node's Reference Set is used to reduce the redundancy between its Activity Set and the Activity Sets of its children node, based on set difference operation. Definition 3.3.7 shows how the reduction process is accomplished.

Definition 3.3.7. *Let* s_i *be the parent node of node* s_j,

$$\begin{aligned} Diff(s_i) &= PS(s_i) - RS(s_i) \quad s_i \, active \, parent \, node. \\ Diff(s_j) &= PS(s_j) - RS(s_i) \end{aligned}$$

are defined to be the 'Difference Sets' of the nodes s_i *and* s_j.

The gathering progresses from the lowest level to the next, all the way to the top level. Each leaf node in the last level of the MNDGT sends its Participation Sets to the parent. The parent node will compute the Reference Set by intersecting all the Participation Sets of the children nodes with its Participation Set. The Difference Sets of the parent node (if it is active) and its children are computed and sent to the Sink. It is possible for any particular node to have two Difference Sets. When a node plays the role of an

active parent, the Difference Set refers to the result of the difference operation between the node's Participation Set (i.e., its Activity Set) and its Reference Set. When a node plays the role of a non-leaf child, the Difference Set refers to the result of the difference operation between the node's Participation Set (i.e., its Reference Set) and the Reference Set of its parent. Leaf and mandatory nodes will have only one Difference Set, and to distinguish between these sets, the notation Diff-RN is used to refer to the Difference Set of a non-leaf child node, and Diff-AN for the is used for the two other cases (the Difference Sets of a leaf child and active parent node).

Table 3.4: Activity Sets

Sensor Id	**Activity set**
s_2	{1, 2, 3, 4, 5}
s_4	{1, 3, 4, 5, 6, 8}
s_5	{1, 3, 4, 5}
s_7	{1, 2, 3, 5, 6, 7, 8}
s_{12}	{1, 2, 3, 4, 6, 8, }
s_{13}	{1, 2, 3, 5, 8}
s_{14}	{1, 3, 4, 5}
s_{16}	{1, 2, 3, 4, 6, 7}

The following example illustrates the In-network Reduction technique in more detail. Table 3.4 shows the Activity Sets of the active nodes of the MNDGT presented in Figure 3.4. Let us start with the leaf nodes s_{13} and s_{14}. Both are leaf nodes and have Participation Sets that are the same as their Activity Sets. Once the parent node s_{10} receives these Participation Sets, it computes the Reference Set by intersecting them together. s_{10} does not participate in computing the reference set because s_{10} is not an active node-Table 3.5 shows the Reference Set of node s_{10}, as well as the Reference Sets of all the other parent nodes-s10 computes the Difference Sets of nodes of s_{13} and s_{14} (i.e., Diff-AN(s_{13}) and Diff-AN(s_{14})) and sends them to the Sink. The same process is carried out for s_{12}, which receives the Participating Set of node s_{16}, and computes the Reference Set by intersecting the Participating Set of node s_{16} with its own. s_{12} then

computes the Diff-AN(s_{16}) and Diff-AN(s_{12}) and sends them to the Sink. The process is then picked up at s_7, which receives the Participation Sets of nodes s_{10} and s_{12}, both of which are non leaf nodes that will participate through their Reference Sets. Node s_7 computes its Reference Set by intersecting the received sets with its Participation Set. It then computes Diff-AN(s_7), Diff-RN(s_{10}), and Diff-RN(s_{12}) and sends theses to the Sink. The process progresses in the same way for all other sensors, level by level. Note that when the Participation Sets are accumulated at the nodes that are one hop away from the Sink, they can be forwarded to the Sink directly. Table 3.6 shows the computed Difference Sets for the nodes in Figure 3.4. Figure 3.6 shows the format of the data message used to send the data to the Sink. The type field is used to indicate the type of the Difference Set, whether Diff-AN or Diff-RN. The reference field identifies the node whose Reference Set will be used to restore the difference carried in the message. Algorithm 4 shows a formal description for the In-Network Data Reduction mechanism.

After receiving all the Difference Sets, the Sink reconstructs the Activity Sets of the active nodes using Lemma 3.3.1.

Lemma 3.3.1. *Let s_i and s_j be active nodes, let s_j be a leaf node, and let s_i be the parent node of the node s_j, then*

$$\begin{aligned} AS(s_i) &= Diff\_AN(s_i) \cup RS(s_i) \\ AS(s_j) &= Diff\_AN(s_j) \cup RS(s_i) \end{aligned}$$

Note that it may be required to reconstruct the Reference Sets before reconstructing the Activity Sets, in which case, Lemma 3.3.2 will be used for reconstruction.

Lemma 3.3.2. *Let s_j be a non leaf node in the MNDGT, and let s_i be the parent node of the node s_j, then*

$$RS(s_j) = Diff\_RN(s_j) \cup RS(s_i)$$

Table 3.5: Reference Sets

Parent Id	**Reference Set**
s_2	{1, 3 }
s_3	{1, 3, 4, 5, 6, 8}
s_7	{1, 3 }
s_{10}	{1, 3, 5}
s_{12}	{1, 2, 3, 4, 6 }

Table 3.6: Difference Sets

Sensor Id	**Diff-AN**	**Diff-RN**
s_2	{2, 4, 5}	
s_3		
s_4	{}	
s_5	{4, 5}	
s_7	{2, 5, 6, 7, 8}	{}
s_{10}		{5}
s_{12}	{8}	{2, 4, 6}
s_{13}	{2, 8}	
s_{14}	{4}	
s_{16}	{7}	

Type	*sender_id*	*Reference*	*Diff_set*

Figure 3.6: Data Message Header

Algorithm 4 In-Network Data Reduction

```
For each s_i ∈ MNDGT
  If s_i active node
    RS(s_i) =(⋂_{∀s_j∈Child(s_i)} PS(s_j)) ∩ PS(s_i);
    Diff(s_i) = PS(s_i) - RS(s_i);
    D_msg=(AN,RS(s_i), Diff(s_i));
    Send(D_msg, P(s_i));
  ELSE
    RS(s_i) =⋂_{∀s_j∈C(s_i)} PS(s_j);
  For eachs_j ∈ C(s_i)
      Diff(s_j) = PS(s_j)- RS(s_i);
      If s_j leaf node
        D_msg=(AN, RS(s_i), Diff(s_i));
      Else
        D_msg=(RN, RS(s_i), Diff(s_i));
      Send(D_msg, P(s_i));
```

3.4 Performance Evaluation

In this section, we present a comparison analysis for the gathering mechanisms that have been proposed in this chapter to collect the data needed for mining Sensor Association Rules. Several experiments have been conducted to compare the performance of the three data gathering mechanisms (Direct Reporting, Distributed Extraction and In-network Reduction). The compression is based on a simulator that has been built using Matlab, version 7.4.0.287 [101]. In this simulator, 1000 sensor nodes have been deployed in a grid of 450m $\times$ 450m. Each sensor is 15m away from any other sensor.

For the Distributed and In-network techniques, the Minimum Nodes Data Gathering Tree (MNDGT) is used for routing data to the Sink. For the Direct technique, a routing tree that encompasses all the nodes in the network is built. TDMA-based MAC protocol is assumed; each node knows the periods in which it can send, and those periods in which to expect data from its children. The rest of the time the node can switch safely to sleep mode. The comparison is based on the number of messages a node may send and the average energy consumption per node. To be able to compare the three mechanisms, we have assumed that each element in the Activity Set for Direct Reporting and Distributed Extraction-Difference Set for In-network Reduction-will be sent in a separate message to the Sink. In reality, a message may contain more than one element, depending on its size. For event generation, we have assumed 50% to 60% of the nodes to be active at

each time slot.

For the mining parameters, we have used a slot size of 60 seconds, historical periods of 5 and 10 days and minimum support values ranging from 10% to 90%. For energy consumption, we have used the first order radio model introduced in [94]. Equation 3.1 shows the energy consumption for sending a k bit message across distance d. E_{elec} is the energy consumption used to run the transmitter and the receiver, which is 50 nJ/bit. E_{amp} = 100 pJ/bit/m^2 is the energy consumption for the transmitter's amplifier. In our experiments, we have assumed that k=480 bits and d=15m. Equation 3.2 shows the energy consumption of turning the receiver circuit. For Distributed Extraction and In-network Reduction, there is an extra cost incurred for maintaining the storage device. We have assumed a Toshiba 16MB NAND flash memory that costs 0.017 uJ to read, write, and erase a byte of data [97, 98]. Table 3.7 summarizes the parameters that have been used in our experiments.

$$E_{TX}(k,d) = E_{elec} * k + \varepsilon_{amp} * k * d^2 \tag{3.1}$$

$$E_{RX}(k) = E_{elec} * k \tag{3.2}$$

Table 3.7: Data Gathering Simulation Parameters

Parameter	**Value**
Number of nodes	1000
Historical period	5, 10, 1-20 days
Flash memory size	16 MB
Minimum support	10% - 90%
Slot size	60 sec
Read, write, and erase from flash	0.017 μJ
Transmission and receive energy	50 nJ/bit
Transmitter's amplifier	100 pJ/bit/m^2
Grid size	450 $\times$ 450 m
Distance between sensors	15 m

Figure 3.7 and 3.8 show the total number of messages versus support values for his-

torical period of 5 and 10 days, respectively. As we can see in these figures, the total number of messages for Direct Reporting is constant for all the support values. This is due to the way that this mechanism reports the Activity Sets: each sensor sends its data without knowing if it will participate in formulating the Sensor Association Rules, and thus the great number of useless messages. For Distributed Extraction, an equal number of messages are generated, in comparison to Direct Reporting at low support values. However, the total number of messages decreases as the support values increase. The decrease in number of messages in Distributed Extraction is due to local decision making by the node based on the cardinality of the local buffer. For In-network Data Reduction, results show that it outperforms both its counterparts by achieving better message reduction than them (the amount of the reduction is 10-30 % of messages achieved by Distributed Extraction and 10-70 % of messages achieved by Direct Reporting). The In-network mechanism shows two areas of optimization to reduce number of messages: (i) the same as in Distributed Extraction, based on the cardinality of sensor buffer; (ii) the removal of some of the redundancy in the Activity Sets. Although we have chosen a sufficient enough density factor for event generation, we will get an approximate reduction using lower density factors. This is explained by having the parent node and its children share the same Reference Set, which is big enough due to the spatial correlation between sensor nodes. Another factor that affects the total number of messages in all three mechanisms is the length of the historical period, as we can see in Figure 3.7 and Figure 3.8. The total number of messages is doubled for the three techniques for the historical period of 10 days, compared to that of 5 days.

For energy consumption, three experiments have been carried out to compute average energy consumption per node over a period of 20 days with minimum supports of 0, 50%, and 90%. Figure 3.9 shows energy consumption per node at a minimum support of 0. For support value 0, both Direct Reporting and Distributed Extraction have the same number of messages. However, the average energy per node for Distributed Extraction is a little higher than that for Direct Reporting, due to the extra energy consumed in

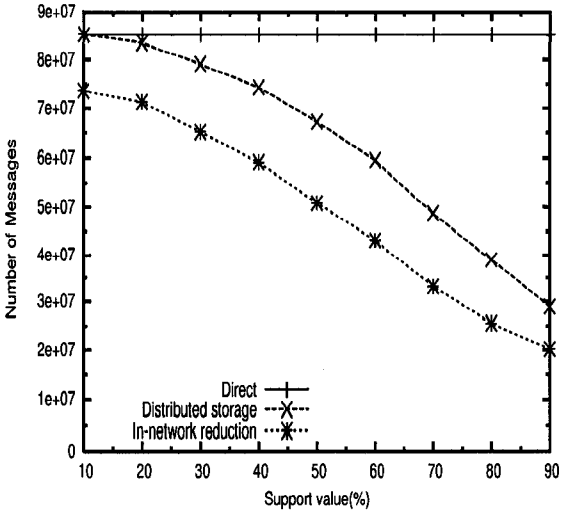


Figure 3.7: Total Number of Messages versus Support Values for 5 Days.

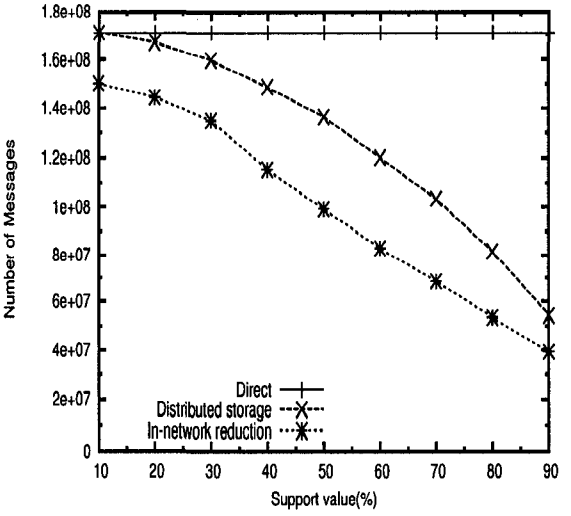


Figure 3.8: Total Number of Messages Versus Support Values for 10 Days.

maintaining the storage attached to the sensor node. The In-network technique still exhibits lower energy consumption compared to both techniques (due to the 'treating' data redundancy). Figures 3.10 and 3.11 show the average energy consumption per node for minimum support to be 50%and 90%. In these figures, Distributed Extraction consumes less energy than Direct Reporting, for a support value of 50%. end even, a better reduction at support value of 90%. In-network Reduction technique outperforms both Direct Reporting and Distributed Extraction for support values of 50% and 90%. Energy reduction ranges between 30% (at a minimum support value of 50%) and 70% (at a maximum support value of 90%) of the average energy consumed by Distributed Extraction; and ranges between 50% (at a minimum support value of 50%) and 90 % (at a maximum support value of 90%) of the average energy consumed by Direct Reporting. Also, it can be seen that the energy consumption for the In-network and Distributed Extraction techniques decreases at a minimum support value of 90%, compared to when the minimum support value is 50% (due to the lower number of active nodes). Another important result that can be deduced from these figures is the amount of energy consumed by each node in In-network Reduction, which ranges between 0 to14 Joules for a support

value of 0; 0 to 9 Joules for a support value of 50%;, and 0 to 2 Joules for a support value of 90%. The amount of energy in current sensor nodes can accommodate the implementation of In-network Reduction technique in wireless sensor network because it consumes a low percentage of the sensors' energy.

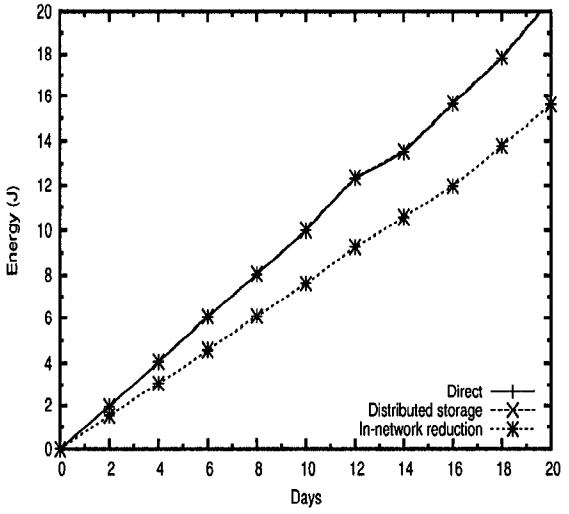


Figure 3.9: Average Energy Consumption Per Node (Min_sup =0%).

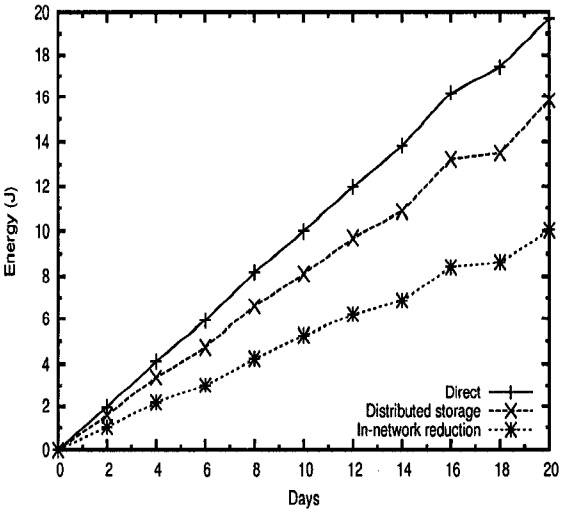


Figure 3.10: Average Energy Consumption Per Node (Min_sup =50%).

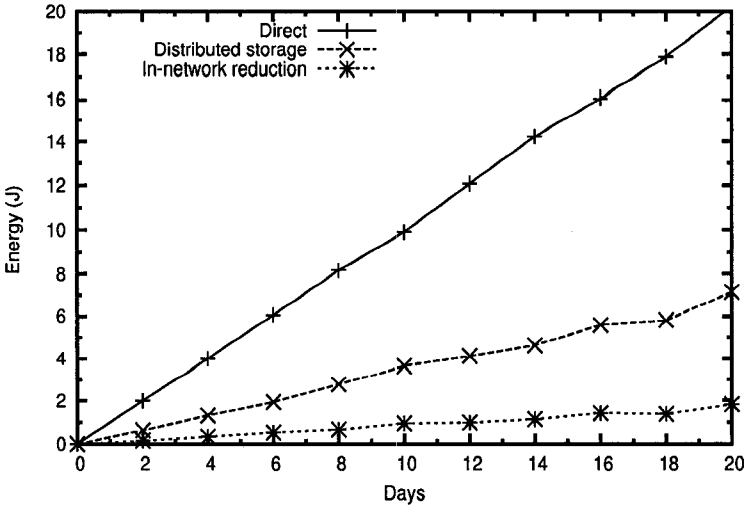


Figure 3.11: Average Energy Consumption Per Node (Min_sup =90%).

3.5 Summary

In this chapter, we have introduced two steps of the Knowledge Discovery Process pertaining to generating Sensor Association Rules. Sensor rules are expected to improve the performance of wireless senor networks by providing the application with rules that capture the temporal correlation between sensor activities. A direct application involves predicting the future source of events, detecting faulty nodes and participating in the resource management process. For preparing the data needed for generating sensor rules, three different data extraction mechanisms have been proposed: Direct Reporting, Distributed Extraction, and In-network Reduction. Performance evaluation have been provided to compare these mechanisms, and the results have shown that the In-network technique achieves message reduction ranging from 10 to 30 % of the messages achieved by Distributed Extraction and 10-70% of the messages achieved by Direct Reporting. In terms of average energy consumption, In-network Reduction achieved reduction ranging between 30% (at a minimum support value of 50%) and 70% (at a minimum support value of 90%) of the average energy consumed by Distributed Extraction; and between 50% (at a minimum support value of 50%) and 90% (at a minimum support value of 90%) of the average energy consumed by Direct Reporting.

Chapter 4

The Positional Lexicographic Tree

4.1 Introduction

In the pervious chapter, we introduced different mechanisms for preparing the data needed for mining Sensor Association Rules. Whichever mechanism is used, a large amount of meta-data (i.e., data about sensor activities) will accumulate in a short time. Moreover, generating the frequent patterns is an exponational problem that requires a high memory space and CPU time [12]. This gives rise to the need for a data structure that maintains and compresses behavioral data and presents it in a way that improves the mining process. Recall that sensor behavioral data is a set of epochs where each epoch is a pair: a time stamp, and a pattern of sensor nodes that detected events within this time stamp.

In this chapter, we propose a data structure which we will refer to as Positional Lexicographic Tree (PLT); it will be used by the Sink node to store and compress the meta-data received from the sensor nodes, using one of the method described in the pervious chapter (Direct Reporting, Distributed Extraction, and In-network Reduction). In addition to its ability to compress data, PLT provides fast access and an easy subset-checking mechanism for the data. These factors are vital for efficient mining of the frequent Sensor Association Patterns. Generating frequent patterns requires exploring

an exponential search space; for example, if we have a set of n sensors, then there are 2^n possible patterns that should be checked against the collected epochs to determine the support count of each pattern. In this perspective, PLT is an essential part of the data preparation and mining steps in the Knowledge Discovery process in wireless sensor networks.

The remainder of this chapter is organized as follow: Section 4.2 defines the basic concepts of Positional Lexicographic Tree; section 4.3 describes the PLT construction process; section 4.4 presents the mining algorithm that will generate the frequent Sensor Association Patterns from the PLT structure; section 4.5 provides the performance evaluation of the PLT structure; and section 4.6 summarizes the chapter.

4.2 Preliminary

Given a database of epoch that stores the meta-data about sensor activities, the simplest, easiest way to generate the frequent Sensor Association Patterns is to consider all the possible subsets that can be formulated from the set of sensors, before scanning the database to determine the support count of each subset discovering the set of the frequent patterns. However, this method is costly and exhausts considerable time and memory.

Position Lexicographic Tree (PLT) assists the Sink in generating the frequent patterns by structuring and indexing the activity data in the behavioral database. The idea behind the PLT is to convert the epochs' patterns in the database to a new form that depends on the lexicographic distance between the sensor identifiers appearing in the same pattern. In this section, we define and present the main concepts of the PLT.

Let $S = \{s_1, s_2, \ldots, s_n\}$ be the set of sensor identifiers in a particular wireless sensor network. A lexicographic order (i.e., dictionary order) is assumed among the sensor identifiers (e.g., s_2 occurs lexicographically after s1). Lexicographic order between elements is represented by a tree structure called the lexicographic prefix tree [15].

Definition 4.2.1. *A lexicographic prefix tree is a tree representation in which the root*

node is labeled null, and all other nodes represent elements in the set S. Each node in the tree is linked to the nodes labeled with the sensor identifiers that occur after the node's identifier in the lexicographic order.

Figure 4.1 illustrates the lexicographic tree of the set $\{s_1, s_2, s_3, s_4\}$.

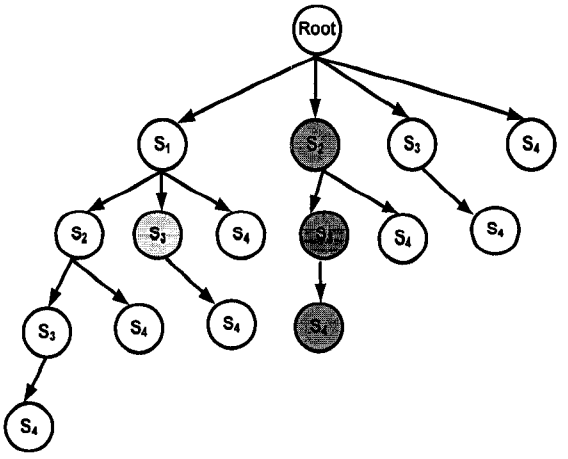


Figure 4.1: The Lexicographic Tree for the Set $\{s_1, s_2, s_3, s_4\}$

Property 4.2.1. *For each subset $S' \subseteq S$, there exists a path, starting from the root node in the S's lexicographic tree, in which there is a one-to-one mapping between the labels of the nodes in this path and the elements in S'.*

As an example of the above property, the path with dark grayed nodes maps the elements of the set $\{s_2, s_3, s_4\}$. Note that the root node is excluded from the path.

Definition 4.2.2. *$Rank(s)$ is a function that maps each sensor node s (where $(s \in S)$) to a unique integer number, so that the lexicographic order is preserved.*

For the sensors in Figure 4.1, we will use the following ranks: $Rank(s_1) = 1$, $Rank(s_2) = 2$, $Rank(s_3) = 3$, and $Rank(s_4) = 4$.

Definition 4.2.3. *pos(s) is defined to be a function that maps each sensor node s, (where ($s \in S$)), in the lexicographic tree to an integer number that represents the lexicographical distance between the node's identifier and its parent identifier.*

Definition 4.2.4. *The Position Lexicographic Tree (PLT) of the set S is defined to be the tree structure constructed from the lexicographic tree of S by replacing the nodes' labels with their position values.*

Definition 4.2.4 sums our logical view of the PLT, which differs from the physical implementation of the PLT. As an example of the PLT, consider the lexicographic tree illustrated in Figure 4.1; the light grayed node with label s_3 has a position value equal to 2 (i.e., the lexicographic distance between this node identifier and its parent identifier is equal to 2); this node will be replaced by a node in the PLT representation that is labeled with a position value equal to 2. Figure 4.2 shows the PLT structure of the set $\{s_1, s_2, s_3, s_4\}$. The Rank function is used to calculate the positions of the nodes as follows: $pos(j) = Rank(j)$ - $Rank(i)$, where j is a child node of node i. $Rank(Root)$ and $pos(Root)$ are set to zero.

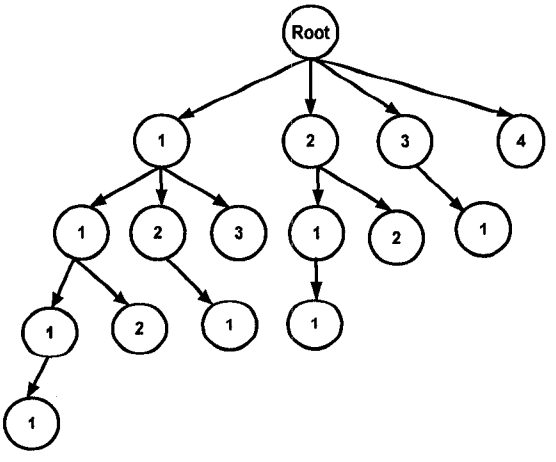


Figure 4.2: The Positional Lexicographic Tree for the Set $\{s_1, s_2, s_3, s_4\}$

Definition 4.2.5. *Let $S = \{ s_1, s_2, \ldots, s_n \}$ be a set of sensor identifiers. $Path(S)$ is defined to be the set of all paths in the PLT from the root node to any other node.*

The elements of $Path(S)$ will be represented by lists. For simplicity, we will exclude the root node from these paths. As an example, consider the PLT portrayed in Figur 4.2; if we omit the value of the root node, then the elements of $Path(S)$ can be listed as follows: $Path(S)$ = {[1], [2], [3], [4], [1, 1], [1, 1, 1], [1, 1, 2], [1, 1, 1, 1], [1, 2], [1, 2, 1], [1, 3], [2, 1], [2, 1, 1], [2, 2], [3, 1]}.

Definition 4.2.6. *Let $P = \{s_1, s_2, \ldots, s_k \}$ be a pattern of sensors, such that $P \subseteq S$. $V(P)$ = [pos(s_1), pos(s_2),...,pos(s_k)], is then defined to be the position vector of the pattern P, such that pos(s_i), $1 \leq i \leq k$ is the position value of the path's node in the lexicographic tree of S that maps s_i in P.*

As an example of the above definition, consider the pattern $P = \{s_2, s_3, s_4\}$ that maps the path with dark grayed nodes in Figure 4.1; V(P) = [2, 1, 1].

The elements of $Path(S)$ represent the position vectors of all the possible subsets of set S. Recall that we are excluding the value of the root nodes from the path elements. Lemma 4.2.1 describes the relationship between the position vectors and the elements of P$ath(S)$.

Lemma 4.2.1. *For each pattern $P \in Power(S)$, $V(P)$ has a unique image in $Path(S)$; where $Power(S)$ represents the power set of S (i.e., all the possible subsets of S).*

Lemma 4.2.2. *Let $P = \{s_1, s_2, \ldots, s_k\}$ be a pattern of sensors, and let $V(P)$ = [pos(s_1), pos(s_2),...,pos(s_k)] be the corresponding position vector of P. For each s_i, 1 <= i <= k, in P, it then holds that*

$$Rank(s_i) = \sum_{j=1}^{i} pos(s_j)$$

Recall that all the above definitions and lemmas are introduced to illustrate our logical view of the position lexicographic tree, which is different from the physical implementation, as we will see in next section.

After defining the basic terminologies that will be used in this chapter, the next section discuses the process of converting the database of epochs into its equivalent PLT representation.

4.3 PLT Construction Process

In this section, we illustrate the process of converting the database of epochs into a Position Lexicographic Tree (PLT) structure. Recall that each epoch $E(E_{ts}, P)$ in the behavioral database consist of two parts: the epoch's time stamp (E_{ts}) and the pattern P of sensors that have detected events within time E_{ts}. We will assume that the first scan of the database is conducted, and the set of frequent sensors are already known and sorted according to the decreasing order of their frequencies.

Definition 4.3.1. *Let $E(E_{ts}, P)$ be a particular epoch in the behavioral database:*

1. *P' is defined to be the set of frequent sensors in P sorted according to the decreasing order of their frequencies. P' is called the frequent pattern of the epoch.*
2. *$V(P')$ is defined to be the position vector of the epoch E.*

Definition 4.3.2. *Given a database (DS) of epochs, a PLT structure of DS is defined to be a set of partitions, such that each partition stores the position vector representations of the epochs' frequent patterns (i.e., $V(P')$) that share the same sensor as the last sensor in these patterns. We refer to each PLT partition by $PLT[i]$, where i is the rank of the shared sensor.*

Each PLT partition has one entry consisting of three fields for each epoch: (i) the vector filed that stores the position values of the epoch's position vector; (ii) the support

field that stores the support count of the epoch in the database; (iii) the comparison value, which is an integer value used as an index to access the vectors (it will be discussed later). Each partition is indexed by the rank of the shared sensor in the epochs' frequent patterns (i.e., the rank of the last sensor in the epochs' frequent patterns). We will illustrate the construction process via a running example using the database presented in Table 4.1, and defining a minimum support(*min_sup*) equal to two epochs.

Table 4.1: Database of Epochs

Time stamp	***Pattern***
1	$s_1 s_2 s_3$
2	$s_1 s_2 s_3$
3	$s_1 s_2 s_3 s_4$
4	$s_1 s_2 s_3 s_4$
5	$s_2 s_3 s_6$
6	$s_1 s_3 s_5$
7	$s_1 s_2$

As noted before, only the frequent sensors can participate in formulating longer frequent patterns. The first step in constructing the PLT is to scan the database in order to determine the set of frequent sensors (i.e., those sensors that have frequencies exceeding the given minimum support value). Given a minimum support equal to 2 epochs, the set of frequent sensors found in Table 4.1 is equal to $\{(s_1, 6), (s_2, 6), (s_3, 6), (s_4, 2)\}$, where ($s_i$, x) represents the sensor's identifier and its support. The sensors are then sorted in descending order according to their frequencies, and ranks are then assigned. As a result, we have: Rank(s_1) =1, Rank(s_2) = 2, Rank(s_3) = 3, Rank(s_4) = 4. After this, another database scan is conducted, and for each epoch the set of infrequent sensors is filtered out. The remaining sensors are sorted according to the descending order of their frequencies (i.e., constructing the frequent pattern of the epoch). A positional vector for these frequent sensors is then created and inserted into the PLT partition that is indexed with the rank of the last sensor in the epochs' frequent pattern. To illustrate,

let us consider epoch number six in Table 4.1; sensor s_5 is filtered out (as it is not frequent), and we are left with the pattern $\{s_1, s_3\}$, which is already sorted according to the descending order of frequencies. Following, the position vector [1, 2] for this pattern is constructed, and inserted in the PLT partition that is indexed with 3 (3 is the rank of the last sensor in the epoch's frequent pattern). Recall that the ranks of the sensors are used to calculate the positional vector, for the epochs in Table 4.1, we have the following set of vectors V_1 = [1, 1, 1], V_2 = [1, 1, 1], V_3 = [1, 1, 1, 1], V_4 = [1, 1, 1, 1], V_5 = [2, 1], V_6 = [1, 2], and V_7 = [1, 1]. V_1, V_2, V_5, and V_6 are stored in PLT[3] (the PLT partition that is indexed with 3), as they share the same sensor as the last sensor in their epochs' frequent patterns, with 3 as the rank of this shared sensor. V3 and V4 are inserted in PLT[4], while V7 is inserted in PLT[2]. Figure 1.3 depicts the PLT representation for the database in Table 4.1.

PLT[4]

pos(1)	pos(2)	pos(3)	pos(4)	Support	Com
1	1	1	1	2	4

PLT[3]

pos(1)	pos(2)	pos(3)	Support	Com
1	2		1	5
2	1		1	5
1	1	1	2	3

PLT[2]

pos(1)	pos(2)	Support	Com
1	1	1	2

Frequent sensors are $\{(s_1, 6), (s_2, 6), (s_3, 6), (s_4, 2)\}$
Ranks are s_1 =1, s_2 =2, s_3=3, s_4=4 .

Figure 4.3: The PLT Structure for the Database in Table 4.1

A design choice that improves the performance of the PLT structure is keeping it in a sorted format, which allows for fast access to the vectors during the mining process and the process of inserting new vectors (that demand checking to know whether the vector is already presented in the partition before inserting it). Sorting can be achieved in two

ways; the first is based on the comparison value of the vector, which is an integer value that captures the distance of the vector from the origin , and is defined as

$$V(P').Com = \sum_{j=1}^{i} pos(s_j)^2$$

where i is the length of the vector. The comparison value of the vector [1, 2, 1] is 6 (i.e, $1^2 + 2^2 + 1^2$), and is used, besides for sorting, as an index for locating the vectors. The second sorting criterion is based on the lexicographical order of the vectors, which is used when the comparison values of the vectors are equal. Algorithm 5 shows a formal description for the PLT construction process.

Algorithm 5 PLT Construction

PLT_Construction(DS, Min_sup)
Fs = {Frequent sensors in DS}.
Sort the elements of Fs according to the descending order of their frequencies.
Assign ranks to FS's elements.
For each $E \in DS$
Let $E' = [s_1, \ldots, s_k]$ the frequent sensors in E sorted according to the descending order of their frequencies.
$V(E') = [pos(s_1), \ldots, pos(s_k)]$.
$V_{E'}.Com = \sum_{i=1}^{k} pos(s_i)^2$.
Index = $\sum_{i=1}^{k} pos(s_i)$.
Insert V(E') to PLT[Index].
End{for}

4.4 The PLT Mining Process

The Position Lexicographic Tree (PLT) represents the data in a structured and compressed format. However, to generate the frequent association patterns, a mining algorithm needs to be developed; one that is able to traverse the PLT structure and determine the frequent patterns, along with their frequencies. Mining algorithms have been categorized into two main approaches: a candidate generation approach [12], and a pattern growth approach [24]. The candidate generation approach scans the database several times and generates the frequent patterns in a level wise manner (i.e., all the patterns

with the same length are generated in the same pass). The process starts by considering a set of potential frequent patterns (i.e., a candidate set) before a scan of the database is conducted, and yields a set of real frequent patterns, which are subsequently used to generate next pass's candidate set. This level wise manner is not seen in the pattern growth approach, which adopts depth wise manner that 'divides and conquers' to generate the frequent patterns (i.e., it divides the databases into several structures and generates the entire frequent pattern presented in one division before considering another). See chapter 2 for more details on these approach.

In this section, we present the mining procedure that efficiently generates the frequent patterns from the PLT structure. The proposed PLT mining process follows the pattern growth approach [24], and considers each PLT partition separately. Each partition goes through several recursive steps that involve building several PLT structures, until all the frequent patterns have been generated. This process is fathomed: if we manage to put together all the epochs in which a particular pattern participate in one structure, we will refer to this structure by the conditional structure of the pattern, then we can generate all the frequent patterns in which this pattern participate without considering any other epochs. In this perspective, the PLT structure constructed from the original database can be seen as the conditional structure for the empty set (i.e., PLT_ϕ), simply referred to as PLT. All the epochs that support the empty set are presented in PLT_ϕ. In general, the notation PLT_P refers to the conditional structure of pattern P.

To generate frequent patterns, we initially consider the frequent sensors presented in the PLT, which are assigned a length equal to one; they will be the seeds from which longer frequent patterns are build. Each pattern's conditional structure is a new PLT structure that stores the epochs that support it; it is constructed from what is called the conditional vectors of the pattern. In what will follow, we start by giving some definitions pertaining to the PLT mining process, and then describe the process in more details using a running example.

Definition 4.4.1. *The conditional vectors of sensor s_i are those vectors constructed by*

removing the last positional value from each vector in $PLT[Rank(s_i)]$.

Definition 4.4.2. *Let PLT_P be the conditional PLT structure for the pattern P, and let* s_i *be one of the sensors in this conditional structure; the conditional structure of the pattern* $P \cup s_i$ *(i.e.,* $PLT\_P \cup s_i$*) is the PLT structure that is built using the patterns constructed from the conditional vectors of* s_i *in* $PLT\_P[Rank(s_i)]$.

The PLT mining process consists of repetitive steps that are the same for all partitions in the original PLT structure and the partitions in the conditional structures that will be produced during the mining process. The partitions are considered in decreasing order of their indices (Recall that each partition's index refers to a particular sensor). The mining process starts with the partition that is indexed with the maximum rank in the original PLT structure. The sensor of this rank is considered to be the initial pattern in the process, since all the epochs that support this sensor are presented in the PLT partition that is indexed with its rank.

Let us assume that the maximum rank is r, then from PLT[r] (i.e., the PLT partition that is indexed with r), we extract the conditional vectors of r, which are those vectors in PLT[r] excluding the last position value from each vector (see definition 4.4.1). The Pattern's conditional vectors represent its projected database, which is the set of patterns in the database that support it [15]. Using the PLT structure presented in Figure 4.3, we start from PLT[4], 4 being the maximum rank presented in this PLT, and extract the conditional vector CV_1 = [1, 1, 1] with support count equal to two. Next, we construct s_4 conditional structure, which is a new PLT structure similar the original PLT, and is built using the same construction algorithm (see definition 4.4.2). Note that s_4 is the first frequent pattern of length one considered in the mining process. The new PLT, which we refer to as PLT_s_4, is s_4's conditional PLT. The new PLT is built from the patterns constructed from its conditional vectors. Lemma 4.2.2 is used to reconstruct the patterns from the conditional vectors. For CV_1 we construct the pattern $\{s_1, s_2, s_3\}$ with support count equal to 2. Note that when building the conditional structures, we go through the same construction process described in the pervious section, consisting of filtering

the infrequent sensors, sorting and assigning ranks (may differ from the actual ranks), and constructing the position vectors. Figure 4.4 shows the conditional PLT of sensor s_4, which has been built using the patterns that were constructed from its conditional vectors. It may happen that the conditional structure has more than one partition, but in our example we just assume one partition.

$PLT\_s_4[3]$

pos(1)	pos(2)	pos(3)	Support	Com
1	1	1	2	3

Frequent sensors are $\{(s_1, 2), (s_2, 2), (s_3, 2)\}$
Ranks are s_1 =1, s_2 =2, s_3=3.

Figure 4.4: s_4's Conditional Structure (PLT_s_4).

Before we proceed further in describing the mining process, we shall illustrate that the PLT structure from which the conditional vectors are extracted should be updated. Each vector, in any PLT partition, supports a number of sensors equal to its length. When we consider one partition for extracting the conditional vectors of a particular sensor, the constructed conditional vectors should be used to update the other PLT's partitions in the same structure. The 'update' process is accomplished by inserting each conditional vector in the partition that is indexed with a value equal to the rank of the last sensor in the pattern representation of the conditional vector. For instance, the conditional vector [1, 1, 1], which has a pattern representation equivalent to $\{s_1, s_2, s_3\}$, should be inserted into PLT[3], 3 being the rank of s_3, 2 being its support count. Also, new compression values are computed for the inserted vectors. Figure 4.5 illustrates this process by showing the original PLT presented in Figure 4.3 after the 'update' process.

The mining process continues by considering PLT_s_4 and proceeds as if it were the original PLT structure. It starts from $PLT\_s_4[3]$ (Figure 4.4), which encodes the vectors containing the sensor with rank 3 (i.e., the maximum rank in s_4's conditional PLT). At this stage, the first frequent pattern $\{s_4, s_3\}$ is produced with a support count equal to 2 (i.e., 2 is the support count of sensor s_3 in the PLT_s_4). The process continues by

PLT[3]

pos(1)	pos(2)	pos(3)	Support	Com
1	2		1	5
2	1		1	5
1	1	1	4	3

PLT[2]

pos(1)	pos(2)	Support	Com
1	1	1	2

Frequent sensors are $\{(s_1, 6), (s_2, 6), (s_3, 6)\}$
Ranks are s_1 =1, s_2 =2, s_3=3.

Figure 4.5: The Updated PLT after Extracting s_4 Conditional Vectors.

building the conditional structure of patterns $\{s_4, s_3\}$ (i.e., $PLT\_s_4s_3$), and at the same time updating PLT_s_4 by the conational vectors of sensor s_3 (extracted from $PLT\_s_4[3]$). Figure 4.6 shows the PLT_s_4 after the 'update' process; figure 4.7 shows the conditional structure of the pattern $\{s_4, s_3\}$.

$PLT\_s_4[2]$

pos(1)	pos(2)	Support	Com
1	1	2	2

Frequent sensors are $\{(s_1, 2), (s_2, 2)\}$
Ranks are s_1 =1, s_2 =2.

Figure 4.6: PLT_s_4 after the 'Update'

$PLT\_s_4s_3[2]$

pos(1)	pos(2)	Support	Com
1	1	2	2

Frequent sensors are $\{(s_1, 2), (s_2, 2)\}$
Ranks are s_1 =1, s_2 =2.

Figure 4.7: $PLT\_s_4s_3$

The mining process then proceeds from $PLT\_s_4s_3$ (Figures 4.7), considers the maximum rank, 2, and generates the frequent pattern $\{s_4, s_3, s_2\}$ with support count equal to 2 (2 is the support count of s_2 in $PLT\_s_4s_3$). Next, the conditional structure of $\{s_4, s_3, s_2\}$ is built (Figure 4.8) and the conditional structure of $\{s_4, s_3\}$ is updated (Figure 4.9).

Moving on with the mining process, the maximum rank in the conditional structure of pattern $\{s_4, s_3, s_2\}$ (i.e., $PLT\_s_4s_3s_2$, Figure 4.8) is considered-equal to one-and frequent pattern $\{s_4, s_3, s_2, s_1\}$, with support count equal to 2, is generated. Since there are no conditional vectors for s_1, and there are no more partitions in $PLT\_s_4s_3s_2$, the mining

$PLT\_s_4\_s_3\_s_2[1]$

pos(1)	Support	Com
1	2	1

Frequent sensors are $\{(s_1, 2)\}$
Ranks are s_1 =1.

Figure 4.8: $PLT\_s_4s_3s_2$

$PLT\_s_4\_s_3[1]$

pos(1)	Support	Com
1	2	1

Frequent sensors are $\{(s_1, 2)\}$
Ranks are s_1 =1.

Figure 4.9: $PLT\_s_4s_3s_2$ after 'Update'

process goes back recursively to consider the next sensor in the $PLT\_s_4s_3$ conditional structure (rank equal to one, Figure 4.9), and generates the frequent pattern $\{s_4, s_3, s_1\}$, with support count equal to 2. Since there are no conditional vectors for s_1, and there are no more partitions in $PLT\_s_4s_3$, the process returns recursively to consider the next frequent rank in s_4's conditional structure (rank equal to two, Figure 4.6). At this point, the frequent pattern $\{s_4, s_2\}$, with support count equal to 2, is generated, and the conditional structure $PLT\_s_4s_2$ is built (Figure 4.10). The PLT_s_4 structure is updated by s_2's conditional vectors-extracted from $PLT\_s_4[2]$ (Figure 4.11).

$PLT\_s_4\_s_2[1]$

pos(1)	Support	Com
1	2	1

Frequent sensors are $\{(s_1, 2)\}$
Ranks are s_1 =1.

Figure 4.10: $PLT\_s_4s_2$

$PLT\_s_4[1]$

pos(1)	Support	Com
1	2	1

Frequent sensors are $\{(s_1, 2)\}$
Ranks are s_1 =1.

Figure 4.11: PLT_s_4 after 'Update'

The mining process then considers the maximum ranked sensor in $PLT\_s_4s_2$ (Figure 4.10), s_1, and generates the patterns $\{s_4, s_2, s_1\}$, with support count equal to 2.

Since there are no conditional vectors for s_1, and there are no more partitions in $PLT\_s_4s_2$ to be considered, the process returns to consider the next partition in PLT_s_4: $PLT\_s_4[1]$ (Figure 4.11), and produces patterns $\{s_4, s_1\}$, with support count equal to 2. At this point, the mining process for PLT_s_4 has finished and returns to the original structure where it will consider the partition that is indexed with 3 (i.e., PLT[3], Figure 4.5).

The process continues by extracting the conditional vectors of s_3 from the original PLT (Figure 4.5); these vectors include: [1], [2], and [1, 1]-support count equal to 1, 1, and 4, respectively. The conditional structure of s_3 is then built from the equivalent patterns of these vectors (Figure 4.12), simultaneously updating the PLT structure (Figure 4.13).

$PLT\_s_3[2]$

pos(1)	pos(2)	Support	Com
2		1	4
1	1	4	2

$PLT\_s_3[1]$

pos(1)	Support	Com
1	1	1

Frequent sensors are $\{(s_1, 5), (s_2, 5\}$
Ranks are $s_1 = 1, s_2 = 2$.

Figure 4.12: The PLT_s_3

PLT[2]

pos(1)	pos(2)	Support	Com
2		1	4
1	1	5	2

PLT[1]

pos(1)	Support	Com
1	1	1

Frequent sensors are $\{(s_1, 6), (s_2, 6)\}$
Ranks are $s_1 = 1, s_2 = 2$.

Figure 4.13: Updated PLT after Extracting the s_3 Conditional Vectors

The mining process continues from the maximum rank in PLT_s_3 (equal to 2, Figure 4.12), generates the conditional vectors of s_2 (i.e., the sensor with rank 2), and produces the pattern $\{s_3, s_2\}$, with a support count equal to 5. PLT_s_3 (Figure 4.14) is updated, and $PLT\_s_3s_2$ is built (i.e., the conditional structure of the pattern $\{s_3, s_2\}$, Figure 4.15). From $PLT\_s_3s_2$, the mining process generates the pattern $\{s_3, s_2, s_1\}$, with a support count equal to 4. Being that there are no more partitions in $PLT\_s_3s_2$, the

next partition in PLT_s_3 (Figure 4.14) is considered; it is indexed with 1, and the pattern $\{s_3, s_1\}$ is generated, with a support count equal to 5. As this is the last partition in PLT_s_3, the process returns to consider the next partition in PLT structure, which is indexed with 2 (Figure 4.13).

$PLT\_s_3[1]$

pos(1)	Support	Com
1	5	1

Frequent sensors are $\{(s_1, 5)\}$
Ranks are $s_1 = 1$.

Figure 4.14: Updated PLT_s_3

$PLT\_s_3s_2[1]$

pos(1)	Support	Com
1	4	1

Frequent sensors are $\{(s_1, 4)$
Ranks are $s_1 = 1$.

Figure 4.15: The $PLT\_s_3s_2$

Continuing with the mining process, the conditional vectors of s_2 are extracted from the original PLT structure (Figure 4.13), i.e., vector [1], with support count of 5. The conditional structure of s_2 is then built (Figure 1.16) and the PLT is updated (Figure 4.16). Proceeding from $PLT\_s_2[1]$, the pattern $\{s_2, s_1\}$ is generated, with support count equal to 5. Since there are no conditional vectors for s_1, and there are no more partitions to consider in $PLT\_s_2[1]$, the process returns recursively to consider the last partition in the original PLT, which is indexed with 1. At this stage, the process just outputs sensor s_1 with support count equal to 6. Table 4.2 shows all the frequent patterns generated form the database presented in Table 4.1. Algorithms 6 and 7 show formal descriptions for the mining and update processes.

$PLT\_s_2[1]$

pos(1)	Support	Com
1	5	1

Frequent sensors are $\{(s_1, 5)\}$
Ranks are $s_1 = 1$.

Figure 4.16: The PLT_s_2

$PLT[1]$

pos(1)	Support	Com
1	6	1

Frequent sensors are $\{(s_1, 6)$
Ranks are $s_1 = 1$.

Figure 4.17: Updated PLT after Extracting s_2's Conditional Vectors

Table 4.2: Frequent Patterns

Patterns	Support
s_4	2
s_3	6
s_2	6
s_1	6
s_4s_3	2
s_4s_2	2
$s_4s_3s_1$	2
$s_4s_2s_1$	2
$s_4s_3s_2s_1$	2
s_3s_2	5
$s_3s_2s_1$	4
s_2s_1	5

Algorithm 6 The Mining Process

```
Main()
  Index = Maximum rank in PLT
  For i = Index downto 1
      PLT_Mining(PLT, i, φ)
  Endfor

PLT_Mining(PLT, i , P)
   CV_i = the conditional vectors of i
   Update(CV_i, PLT)
   CE = the set of patterns constructed from CV_i
   P=P ∪ "i"
   Output(P)
   PLT_P = PLT_Construction(CE, Min_Sup)
   Cindex = Maximum rank in PLT_P
   For j = Cindex downto 1
        PLT_Mining(PLT_P, j, P)
   End{for}
```

Algorithm 7 The Update Process

```
Update(CV_i, PLT)
   For each Vector v in CV_i
      k = v.Length
      v.Sum = Σ_{j=1}^{k} pos(s_j)
      v.Com = Σ_{j=1}^{k} pos(s_j)^2
      Insert(v, PLT[v.Sum])
   End{for}
```

4.5 Performance Evaluation

In this section, we present the performance evaluation of the Position Lexicographic Tree (PLT) proposed in this chapter. PLT stores and compresses behavioral data collected

from sensor nodes, and presents that data in a structural format that leads to an efficient mining of Sensor Association Rules; it uses the pattern growth approach to generate the frequent patterns. In order to evaluate the performance of the PLT structure, we have compared its performance with the FP-Growth algorithm [24], which is implemented by Bart Goethals [31](It follows the pattern growth approach in mining the frequent patterns). The comparison was based on CPU time and memory usage of the programs in generating the frequent patterns.

An Intel 3.00 GHz Pentium 4 with 2 GB of memory was used in our experiments. Both implementations, PLT and FP-Growth, were coded using C++ compiled with g++ version 3.4.4. We used both (time) and (memusge) commands, available in Linux operating system, in order to measure CPU time and memory size consumed by the PLT and FP-Growth programs. Synthetic data generated by a simulator (that we have built using Matlab version 7.4.0.287 [101]) was used for the comparison. In that simulator, we abstracted the underlying communication protocols, and assumed a reliable delivery of the messages, a task that can be easily accomplished by acknowledging the number of set bits in each sensor's buffer. The simulator's parameters consist of the number of nodes, historical period, and slot size. In our experiments, we used 100 nodes, 5 and 10 days for the historical period, and 60 seconds for the slot size. We assumed that events detection in each sensor is uniformly distributed over the given number of epochs. However, a density factor was assumed to exist between sensor nodes: 3% achieved at minimum support 90% (i.e., certain nodes, equal to 3% of number of nodes, detect events in 90% of the epoch), 6% achieved at 80%, and 9% achieved at minimum support 70%.

For our comparison, we did not only use data generated by the simulator; real data extracted from a sensor network deployed at Intel Berkeley Research lab [102], which has been widely used by the research community [99, 74], was also used. The real data consists of tuples from environmental sensors of a network consisting of 54 sensors, where each sensor reports readings every 30 seconds. A missed reading from a sensor node was considered an undetected event (i.e., each sensor generates an event every 30 seconds;

if the event is successfully delivered to the Sink, the sensor is considered in the epoch). We found this scenario to be helpful in generating patterns regarding lost readings. The measurements from the real data were extracted using filtering routines implemented in MATLAB. Table 4.3 summarizes the characteristics of the data sets that were used in the evaluation process. For all the experiments, we used support values ranging from 10% to 90% of number of epochs.

Table 4.3: Data Sets Properties

Data set	**# of Sensors**	**Historical period (day)**	**Slot Size (sec)**	**Nature**
5-Day	54	5	30	Real
10-Day	54	10	30	Real
S100H5dZ60s	100	5	60	synthetic
S100H10dZ60s	100	10	60	synthetic

Figures 4.18 to 4.21 show the CPU time for the PLT and FP-Growth algorithms during the mining process (using the data sets presented in Table 4.3). For the sake of readability of the graphs, we have omitted part of the results. For synthetic data at low minimum support values, PLT needed 40% to 50% of the CPU time that was required by the FP-Growth. However, the magnitude of reduction decreased when the support value increased (i.e., CPU time for the PLT will be close to the CPU time of FP-Growth, but less). This is due to the lower number of frequent patterns at higher support values. As a result, we can forecast a great reduction in CPU time for high-density data sets at lower support values. The same conclusion applies to the real data sets; though the number of time slots is greater (lower slot size time), CPU time for real data is still less than in the case of synthetic data due to the higher number of nodes and higher density factor (used to generate the synthetic data); this is reflected in the high rate of missed reading in real data.

Memory exhausted during the mining program is another important metric for evaluating the performance of any algorithm that is designed to generate frequent patterns. In order to 'quantize' used memory, we used the (memusage) command provided by Linux

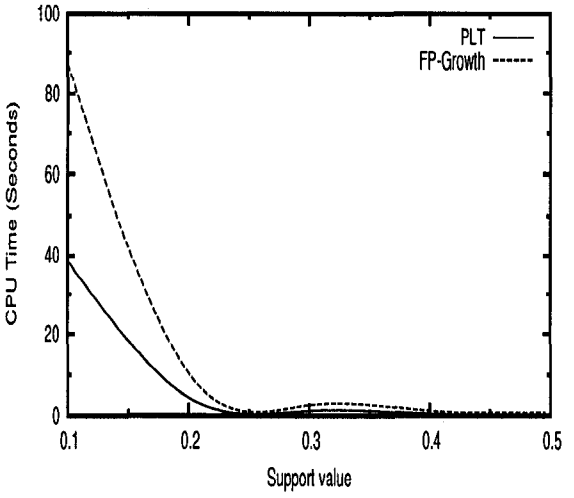


Figure 4.18: CPU Time vs Support Values for S100H5dZ60s.

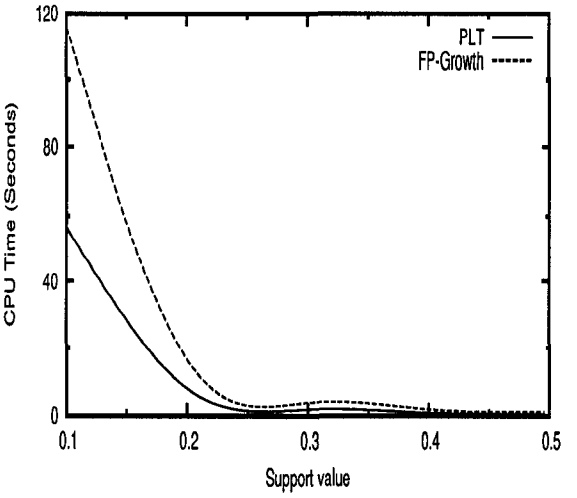


Figure 4.19: CPU Time vs Support Values for S100H10dZ60s.

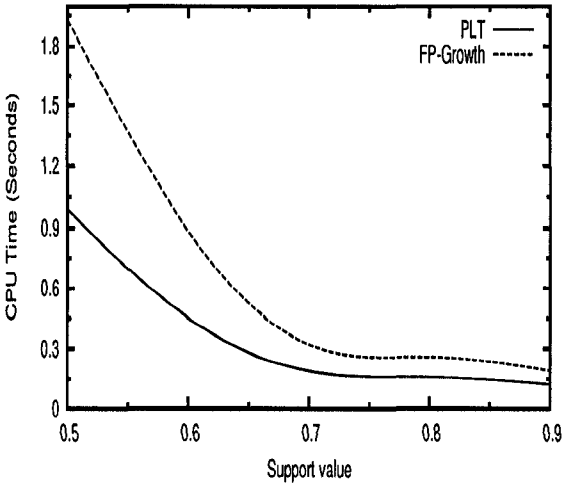


Figure 4.20: CPU Time vs Support Values for 5-day Data.

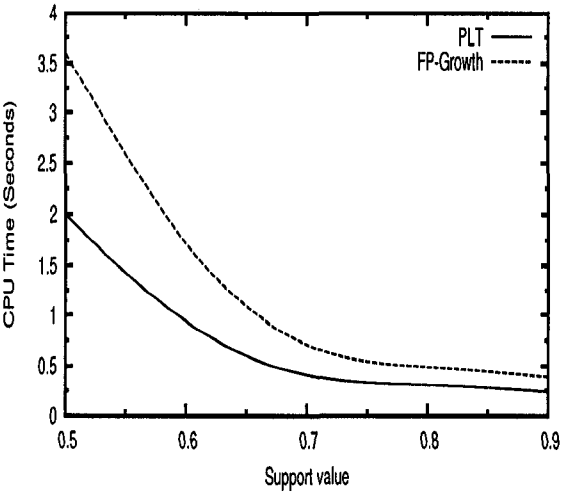


Figure 4.21: CPU Time vs Support Values for 10-day Data.

operating system; memusage 'measures' the memory used by the running program. We used the synthetic and the real data described in Table 4.3 as basis for comparison. All results were computed in Mega Byte. Figures 4.22 to 4.25 show the memory usage, at different support values and using different data sets, for PLT and FP-Growth during

the mining process. All the results show that the mining process using the PLT structure outperforms the FP-growth in terms of memory usage, for all the support values used. For all the data sets, PLT memory consumption fell in the range 30%-40% of memory used by FP-Growth. All figures show that the best reduction was achieved at low support values. However, this reduction in the memory usage decreased when the support value increased. As the case in CPU time, this is due to the lower number of frequent patterns that can be generated at higher support values. In all data sets used in the experiments, the amount of memory used fell in the range 100 and 15000 MB, a reasonable number for contemporary computers.

The graphically-depicted results above entail a study of the properties of the PLT structure that allow the mining process to outperform its counterpart in FP-Growth; these can be summarized as follows:

- The partitioning mechanism used in PLT facilitates the task of locating the conditional vectors of a particular pattern, as opposed to following the nodes' link as in the FP-tree.

- PLT partitions are independent; the entire structure need not reside in the main memory, in contrast to the FP-tree that requires the whole structure to be present in the main memory.

- PLT structure uses the comparison values of position vectors to locate and insert vectors, thus accelerating the process of accessing the PLT structure, whereas the FP-tree does not include such values.

- PLT requires smaller variable sizes for storing positional values, as it uses the lexicographic distance to represent the sensor identifiers, unlike the FP-tree that uses the actual sensor identifiers.

Although the obtained results exhibit reasonable CPU time and memory usage, that will not always be the case; many factors affect CPU time and memory usage:

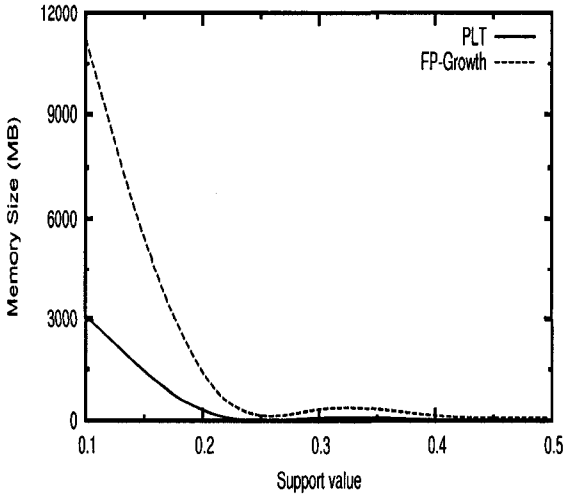


Figure 4.22: Memory Usage vs Support Values for S100H5dZ60s Data.

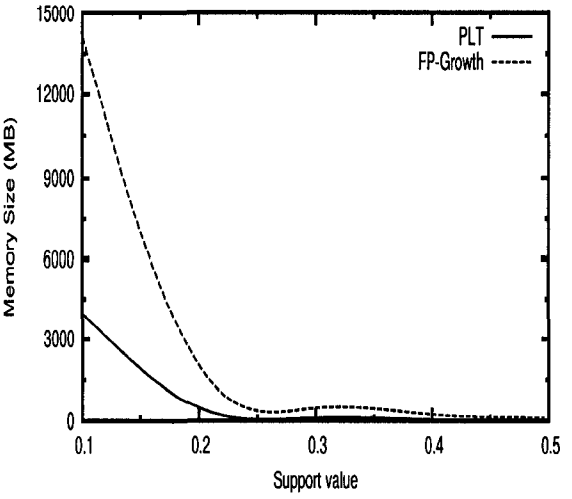


Figure 4.23: Memory Usage vs Support Values for S100H10dZ60s Data.

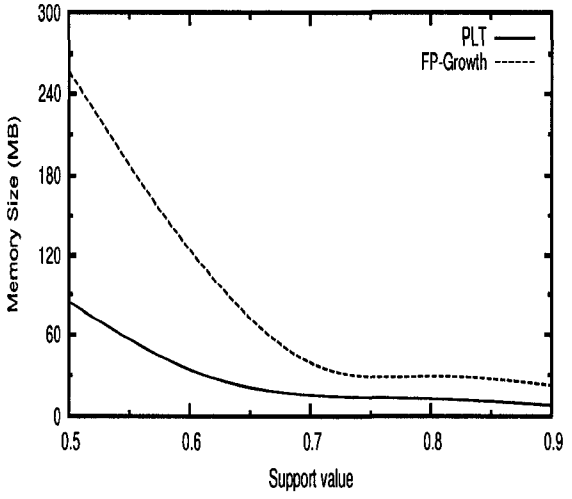


Figure 4.24: Memory Usage vs Support Values for 5-day Data.

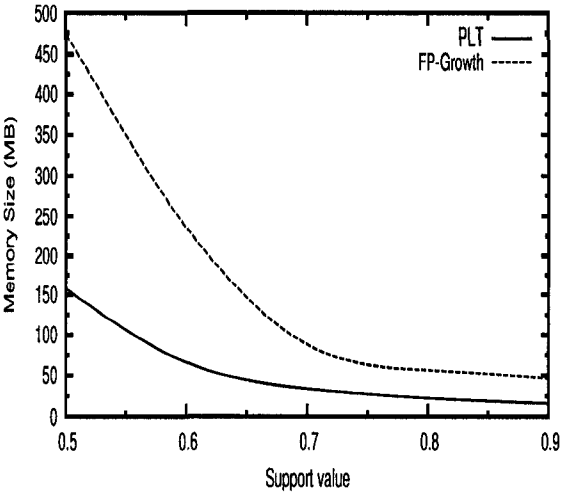


Figure 4.25: Memory Usage vs Support Values for 10-day Data.

- Number of nodes.
- The activity of the network, which affects the density of the data.
- Slot size.

- Historical period.

In the case that any of the above factors combine to generate large volumes of data that the PLT structure may or may not be able to handle admirably, we are certain that we can achieve good performance results compared with other techniques.

4.6 Summary

Generating Sensor Association Rules requires collecting large volumes of meta-data that describe the activity of sensor nodes. Without an efficient mechanism to 'treat' this data, memory and CPU time will be needlessly exhausted to generate the frequent patterns. Throughout this chapter, we have introduced the Positional Lexicographic Tree (PLT) as the main tool to compress and present the meta-data in such a way as to accelerate the mining process and minimize the memory usage. PLT can be seen as an essential part of data preparation and data mining steps in the Knowledge Discovery Process for wireless sensor networks; it uses the lexicographic distance between the sensor identifiers that appear in the same pattern to compress the data, and follows the pattern growth approach for generating the frequent patterns. To evaluate the performance of the PLT, a comparison analysis against FP-Growth has been conducted. The latter is a well known data mining technique that follows the pattern growth approach. Results showed that at low support values, PLT exhausts between 40% to 50% of CPU time used by FP-growth, and 30% to 40% of memory.

Chapter 5

Coverage Based Association Rules

5.1 Introduction

Sensor Association Rules, which was introduced in the pervious chapters, are designed to capture the temporal correlations between sensor activities in Wireless Sensor Networks (WSN), and provide the application with a set of behavioral patterns about sensor nodes. These rules are useful in that they can be used to enhance network performance, and thus its quality of services (see chapter 1 for real scenarios of Sensor Association Rules applications).

Sensor rules discover the correlation between all sensor nodes in the wireless sensor network, regardless of their locations. However, in most applications, a specific location may be covered by many sensors. Moreover, a k-coverage area, $k > 1$ is one of the desired Quality of Services (QoS) properties for WSNs [70]. In a k-coverage sensor network, all k nodes within a specific area are expected to detect the same event; as a result, all k sensor nodes will have the same activity sets during the process of profiling their behaviors when preparing the data needed for generating Sensor Association Patterns. Recall that the activity set of a sensor node consists of time slot numbers in which the sensor has detected events within a given historical period. Generating sensor rules, in a k-coverage network-from all the nodes in the network-results in producing patterns that

are already known by the application through the coverage property of the network.

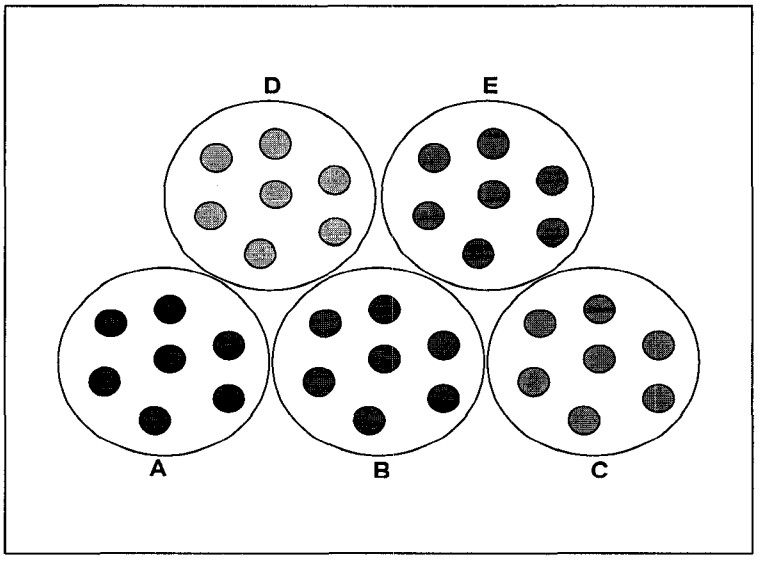


Figure 5.1: Coverage-based Association Rules

Figure 5.1 shows a sensor network that is divided into 5 areas: A, B, C, D and E; each area is covered by seven sensors. In this diagram, all nodes within the same location will be expected to share the same activity sets, since the same event is detected by all sensors in the area. Although the figure shows that the sensors are conveniently separated, their sensing coverage overlaps. However, each sensor will be responsible for the events in its area [68, 69].

In this chapter, we propose a relaxation version of Sensor Association Rules that discovers the correlation between a set of locations (areas) rather than individual sensor nodes. We refer to the new proposed rules by Coverage-based Sensor Association Rules. The proposed rules will be based on partitioning the network into a set of areas, each cover by k-sensor nodes, $k >= 1$. However, we do not assume that k is equal for all areas covered by this network, as it is difficult to guarantee a symmetric coverage, especially at the edges of the network [68].

This chapter is organized as follows: Section 5.2 provides a formal definition for

Coverage-based Sensor Association Rules. Section 5.3 includes simulation results prepared to evaluate the WSN's performance during the processing of data preparation for Coverage-based Rules and Sensor Association Rules. Section 5.4 summarizes the chapter.

5.2 Coverage Based Rules- Formal Definition

In this section, we present a formal definition for Coverage-based Association Rules. We will use the same parameters used in defining Sensor Association Rules, which was presented in Chapter 3. The parameters used include: slot size (λ), historical period (T_his) and minimum support. Time is divided into a set of time slots, of size λ each. Also, we assume that the network uses the same buffering mechanism, used in the In-network Reduction and Distributed Extraction mechanisms, to profile network activities during the given historical period. Coverage-based Association Rules is a buffering mechanism where each senor maintains a buffer of size (T_his/λ). If the sensor detects an event within this time slot, a buffer entry corresponding to the current time slot is set. The network is divided into a set of locations, where each location is covered by k-sensors. Each location has a distinguished node known as the **location manager**. The location manager is responsible for collecting all the activity sets from other nodes within its coverage area at the end of the historical period. Recall that the activity set of a node s, $AS(s) = \{i_1, i_2, ..., i_m\}$, is the set of time slot numbers in which the sensor detected events (i.e., the set of set entries in the sensor's buffer).

Definition 5.2.1. *Let $L = \{l_1, l_2, ..., l_n\}$ be the set of locations in a particular sensor network, and let $C(l_i) = \{s_1, s_2, ..., s_k\}$, $1 <= i <= n$, be the set of sensors covering location l_i . The activity set of l_i, $AS(l_i)$, is defined by as the set produced by the union of all the activity sets of the nodes in $C(l_i)$:*

$$AS(l_i) = \bigcup_{\forall s_j \in C(l_i)} AS(s_j)$$

Definition 5.2.1. *A Location Database (DL) is defined as the set of locations covered by a particular sensor network, along with their activity sets.*

Table 5.1 shows an example of a location database.

Table 5.1: Location Database (DL)

Location	Activity set
l_1	$\{1,3,4,6\}$
l_2	$\{2,4,6\}$
l_3	$\{1,3,5,6\}$
l_4	$\{2,4,5,6\}$

Definition 5.2.2. $P = \{l_1, l_2, ..., l_m\}$, *such that* $P \subseteq L$, *is a pattern of locations.*

Definition 5.2.3. *The support of pattern* $P = \{l_1, l_2, ..., l_m\}$ *in DL is defined by the cardinality of the set that is produced by intersecting all the activity sets of the locations in this pattern.*

$$Support(P) = |\bigcap_{\forall l_j \in P} AS(l_j)|$$

Definition 5.2.4. *A pattern P is said to be frequent patterns if it support greater or equal a given minimum support.*

Definition 5.2.5. *Coverage-based Rule is defined by the implication* $P' \Rightarrow P''$ *where* $P' \subset L$, $P'' \subset L$, *and* $P' \cap P'' = \phi$.

Definition 5.2.6. *The support of the rule* $(P' \Rightarrow P'')$ *is defined by the support of the pattern* $(P' \cup P'')$ *in DL, while the confidence of the rule is defined by:*

$$\textbf{\textit{Conf}}(P' \Rightarrow P'') = \textbf{\textit{Support}}(P' \cup P'') \; / \; \textbf{\textit{Support}}(P').$$

Knowledge Discovery for Coverage-based Rules is the process of generating all rules in the location database that meet an application's pre-defined minimum support and confidence percentage.

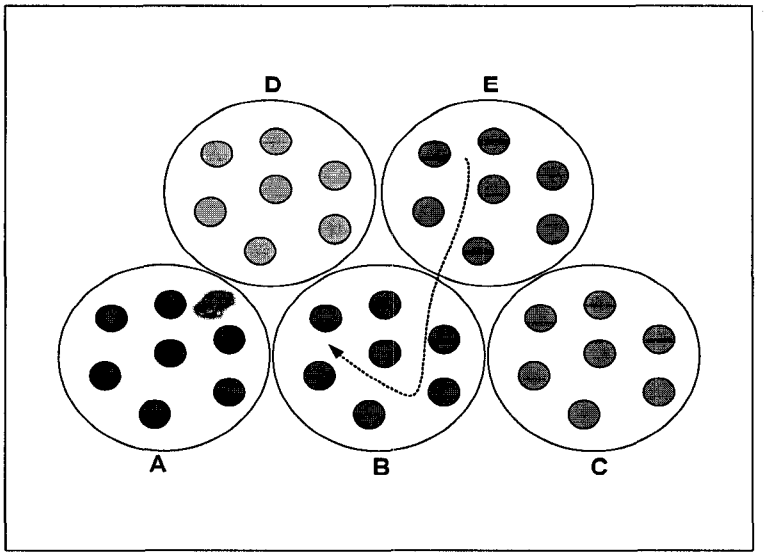


Figure 5.2: Coverage-based Association Rules

An interesting application for Coverage-based Rules sees them predict the location of moving objects. For example, consider the network in Figure 5.2; this network is designed for detecting moving vehicles and reporting their locations to the Sink. Assuming that the Sink possesses a coverage rule that stipulates: (E,B $\Rightarrow$ A) that meets the application's minimum support and confidence percentage, once the it receives reporting messages from the location managers in location E and B, it can easily predict that the next location of the moving vehicle will be A. In this case, the Sink can inform the application about the next expected location of the moving object, a piece of information that proves helpful in many situations.

Generating Coverage-based Rules involve preparing the database that consists of a set of locations in the wireless sensor network and their activity sets. The key to preparing

this database is to partition the network into a set of locations, and to identify the set of sensor nodes that cover each location. Several algorithms have been proposed to address the coverage problem in wireless sensor networks. These algorithms focus on "how well an area is monitored by the sensor networks" [71]. Another class of algorithms, proposed in the domain of coverage, investigates to the possibility of partitioning the network into a set of clusters, such that the area covered by a particular sensor is covered by at least another sensor in the same cluster [68], or by the union of the areas covered by several sensors in the same cluster [69]. These algorithms aim to generate a schedule that identifies which sensor nodes can switch to sleep mode without affecting the coverage property of the network.

The process of preparing data needed for generating Coverage-based Rules (i.e., the location database) can be integrated with any of the coverage-based clustering algorithms. Each coverage-based clustering algorithm consists of three phases [68, 69]: (i) cluster formulation, where the network is partitioned into a set of clusters; (ii) sponsor sets (capable of covering the whole cluster) identification for each cluster, (iii) duty scheduling to identify the sponsor set that should be active in each round. In this setting, the data needed to generate the Coverage-based Rules can be prepared by taking the union of the activity sets of the sponsor nodes at the cluster head. However, for the simulation purpose we will use a fixed partitioning made by the Sink. Recall that we are not proposing a new coverage-based clustering algorithm, as it is out of the scope of this thesis and requires a deep analysis of its efficiency. We are going to provide a basic clustering scheme based on the sensing coverage of the sensor nodes in order to verify the basic concepts of our proposed Coverage-based Association Rules.

5.3 Performance Evaluation

In this section, we present a comparison analysis for the performance of wireless sensor networks during the data preparation process for Coverage-based and Sensor Association

Rules schemes. For simplicity, we will refer to the scheme that generates the association rules between all the sensor nodes as the "All-rules scheme." The metrics used to evaluate the performance of the network are: the number of messages needed to report the activity sets to the Sink, and the average energy consumptions per sensor node. The metrics are collected based on a simulator that has been built using Matlab version 7.4.0.287 [101]. The nodes are assumed to be deployed in a grid environment with a distance of 8 meters between sensor nodes and a sensing range up to 16 meters. Events generation, by each sensor node, is assumed to be uniformly distributed over the possible number of epochs. Sensor nodes covering the same area have the same activity sets.

Although Coverage-based Rules can be implemented on top of any of the coverage-based clustering algorithms introduced in [68, 69], we have implemented it on top of fixed network partitions, where the location managers have been decided by the Sink node, based on the network topology. The number of location mangers depends on the sensing range and the density of the network. If the network is dense enough, and the nodes have long sensing range, then the number of partitions is equal to N/k, where N is the number of sensors in the network and k is the required coverage property. In reality, however, we may have more than this number, as it is hard to guarantee full k-coverage networks, especially at the borders of the network [68]. To cluster the network, we assume that all the nodes have the same sensing range (R) and transmission rang (T), where $T >= R$. N(x) is the set of x's neighbors within radius r, where r = (R/2). Each location manager sends a joint message to all nodes in its neighbor set. A node waits until it receives all the possible joint messages before it joins the closet cluster.

The partitioning is done in such a way that all the areas in the network are covered. The coverage property ranges between 2-4 nodes at the border of the network, and 6 nodes at the central of the network. Each node uses its full sensing coverage for detecting events. Events outside a radius of (R/2) of the location manager (i.e., r)-8 m in our setting-will be discarded. Figure 5.3 shows the sensing range of the sensor nodes.

For the All-rules scheme, we have used the Distributed Extraction mechanism, which

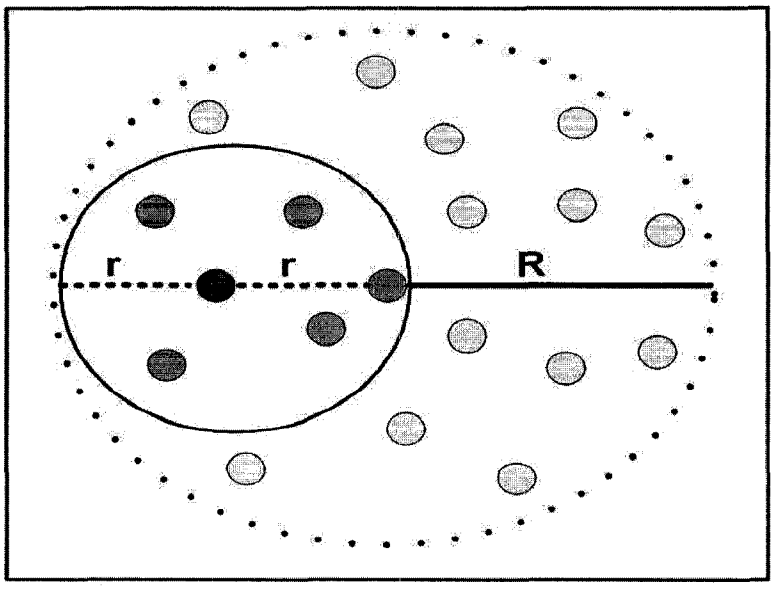


Figure 5.3: Coverage-based Association Rules

was introduced in chapter 3, to prepare the data needed for generating the association rules. In both schemes, the nodes use the buffering mechanism to profile their activities during the given historical period. To route the activity sets to the Sink, a multi-hops tree has been constructed based on the minimum distance to the Sink. The radio and the storage models used by the sensor nodes are the same as those used in Distributed Extraction (see chapter 3 for more details of these models). Table 5.2 summarizes the parameters used in our experiments.

Different experiments were conducted to collect the desired metrics. Figure 5.4 and figure 5.5 show the total number of messages needed to report the nodes' activity sets, for the All-rules and the Coverage-based Rules schemes, for support values ranging between 10%to 90%, and number of nodes equal to 500 and 1000, respectively. Results show that Coverage-based Rules requires fewer messages to extract data than the All-rules scheme; in numbers, and for the given simulation parameters, Coverage-based Rules requires 7%-10% of the number of messages needed for preparing the behavioral data needed for All-rules. This percentage, however, depends on the coverage property of the network.

Table 5.2: Simulation's Parameters

Parameter	*Value*
Number of nodes	500, 1000
Historical period	1-10 days
Flash memory size	16 MB
Minimum support	10% - 90%
Slot size	30 sec
Message size	60 Byte
Read, write, and erase from flash	0.017 μJ/Byte
Transmission and receive energy	50 nJ/bit
Transmitter's amplifier	100 pJ/bit/m^2
Grid size	(185 x 185 m), (248m * 248m)
Sensing range	16 m
Transmission range	16 m
Distance between sensors	8 m
Coverage property	1-6 nodes

The higher the degree of clustering, the fewer is the volume of data to be routed to the Sink. In the worst case (k = 1), the number of messages for the Coverage-based Rules will be equal to that needed to prepare the data for the All-rules scheme, in addition to a few number of control messages needed in the attempt for clustering the network. If Coverage-based Rules is built on top of an already coverage-based clustering algorithm, then this number can be neglected.

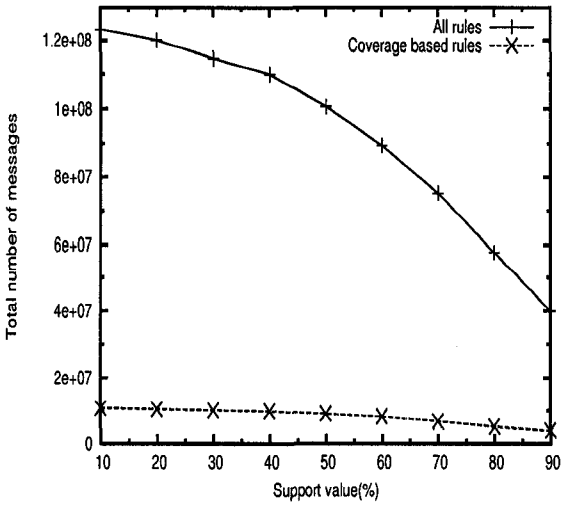


Figure 5.4: Total Number of Messages versus Support Values, for 500 Nodes.

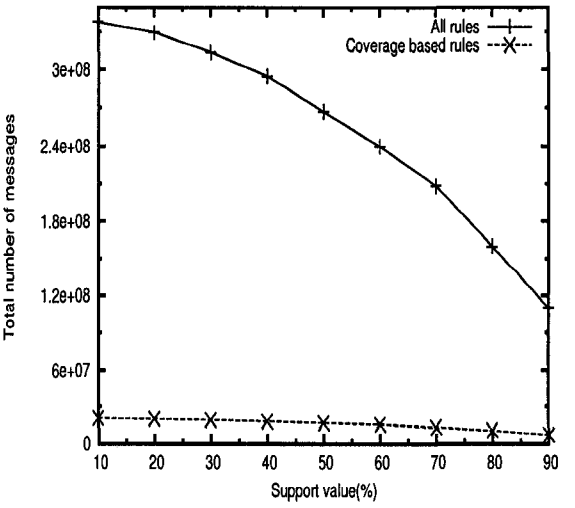


Figure 5.5: Total Number of Messages versus Support Values, for 1000 Nodes.

Another interesting metric about network performance during the process of preparing the data for All-rules and Coverage-based Rules is the average energy consumption per node. We considered two main sources for energy consumption: that needed for maintaining the buffer storage at each node (i.e., the energy consumption for read, write and erase a buffer entry), and the energy exhausted by message transmission. For the latter, we have used the first order radio model [94] that was described in chapter 3. For the sensor storage, we have assumed NAND flash memory from Toshiba. Storage energy consumption parameters (i.e., the energy required to read, write and erase a byte) are listed in Table 5.2 [97, 98].

We have compared the average energy consumption per node of network topologies of 500 and 1000 nodes, and historical period ranges of 1 to 10 days. Figures 5.6, 5.7 and 5.8 show the energy consumption per node at minimum support values of 0, 50% and 90% respectively.

All figures show that the energy consumption per node for the Coverage-based Rules is less than the average energy consumption using the All-rules scheme; in Coverage-based Rules, each node requires anywhere between 50% to 55% of the energy exhausted by a node in the All-rules scheme. The percentages will vary if the number of messages changed. However, there is extra energy consumption to be accounted for in case of Coverage-based Rules, due to the transmission of activity sets from sensor nodes to the location manager.

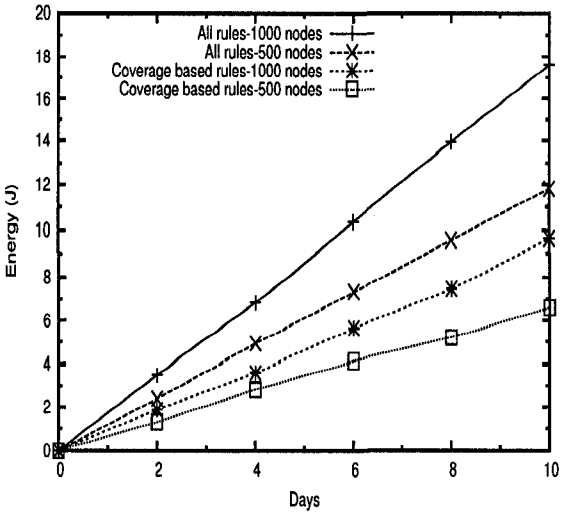


Figure 5.6: Average Energy Consumption, per Node (Min sup =0), for1000 and 500 Nodes.

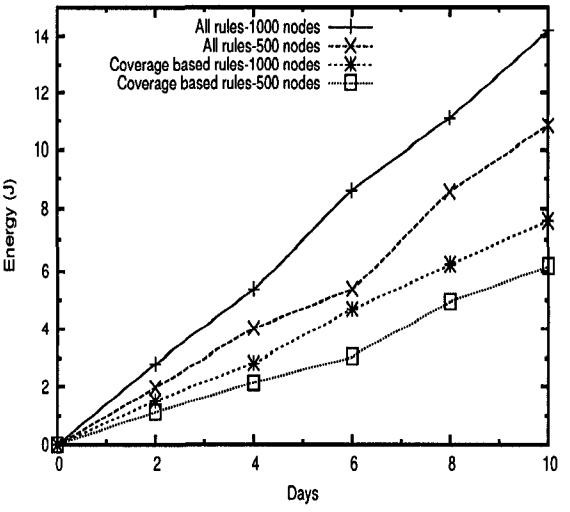


Figure 5.7: Average Energy Consumption, per Node (Min sup =50%), for 1000 and 500 Nodes.

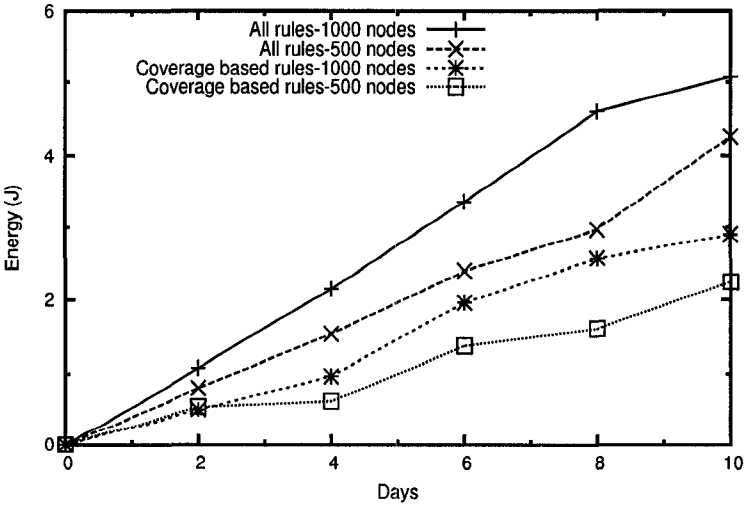


Figure 5.8: Average Energy Consumption, per Node (Min sup =90%), for 1000 and 500 Nodes.

5.4 Summary

Coverage-based Association Rules is the Knowledge Discovery technique, designed specifically for wireless networks, that guarantees a k-coverage property for the network. It is assumed that the network is divided into a set of locations, each covered by k-sensor nodes. In contrast to Sensor Association Rules, Coverage-based Rules discovers the correlation between the set of locations in the network, instead of the correlation between all nodes. The activity set of each location is prepared from the activity sets of the sensor nodes covering the area. To report the performance of Coverage-based Rules, several experiments have been conducted to evaluate the performance of the network during the process of preparing the data needed for generating All-rules and Coverage-based Rules. Results show that Coverage-based Rules require 7%-10% of the number of messages used by the All-rules scheme. In terms of average energy consumption, each node in Coverage-based Rules requires anywhere in the range 50% to 55% of the energy exhausted by a node in All-rules scheme.

Chapter 6

The Chronological Patterns

6.1 Introduction

Knowledge discovery in wireless sensor networks aims to discover several types of patterns: patterns regarding the behavior of the network, and patterns pertaining to the monitored environment. In the pervious chapters, we have introduced two types of behavioral patterns: Sensor Association Rules, which capture the temporal correlations between sensor activities, and Coverage-based Rules, which divide the network into a set of areas and discovers the association between these areas. In this chapter we introduce another type of behavioral patterns, Chronological Patterns, which captures the temporal correlation between sensor activities in detecting events, with a major focus on the order of event detection. For instance, ($[s_1 s_3 s_6]$, λ, 90%) is a chronological pattern that reads: sensors s_1, s_3, and s_6 detected events in the same time interval λ and in the order s_1, s_2, and s_6, respectively, with frequency of pattern equal to 90%. Chronological patterns play an important role in enhancing the performance and Quality of Service (QoS) of wireless sensor networks; for instance, they can be used to compensate for the problem of receiving events out of order in WSNs. Refer to chapter one for a real scenario about the application of chronological patterns.

Although the structure of chronological patterns is similar to that of association

patterns, there are many differences in the way these patterns are generated; the main difference stems from the importance of the order of sensors in the chronological patterns: they must appear in the pattern according to the order in which the events were detected. The difference between chronological patterns and association patterns transforms the issue of support (frequency) counting from subset checking into subsequence checking, and renders the process of generating chronological patterns a combinatorial problem, rather than an exponantional one. In addition to the 'sensor order issue', the techniques used to prepare data needed by the mining process in chronological patters differ from those used in association patterns; in the former, the techniques are implemented on top of the event ordering algorithms.

In this chapter, we present a formal definition of chronological patterns, and introduce a data preparation mechanism to 'process' the behavioral data needed for generating these patterns. Also in this chapter, we propose a new data structure, which we call Chronological Tree (CT); driven by the need for a compressed data structure to store large volumes of behavioral data, CT serves to store data prepared for the mining process. Finally in this chapter, we outlay a mining algorithm that generates the chronological patterns from the Chronological Tree.

The remainder of this chapter is organized as follows: Section 6.2 presents a formal definition of Chronological Patterns; section 6.3 explains our proposed data preparation mechanism for collecting the data needed for generating the chronological patterns; section 6.4 describes the Chronological Tree and its mining algorithm; section 6.5 provides a performance evaluation for the CT and its mining algorithm; and section 6.6 summarizes the chapter.

6.2 Chronological Patterns- Formal Definition

As with defining association patterns, the definition of chronological patterns is inspired by the definition of sequential patterns (proposed in the domain of transactional

databases [32]).

Definition 6.2.1. *Let S be a set of sensors in a particular sensor network. A list $L = [(s_1, t_1), (s_2, t_2), \cdot\cdot, (s_m, t_m)]$ is defined to be a list of pairs, such that $s_i \in S$ and $t_i <= t_{i+1}$ for all $1 <= i < m$.*

The pair (s_i, t_i) encodes the fact that s_i detected an event at time t_i.

Definition 6.2.2. *A chronological pattern $O(L) = [s_1, s_2, \cdot\cdot, s_m]$ of a list L is defined to be the list of the sensors presented in L sorted in increasing order of their timestamps (i.e., the list L without the associated timestamps).*

Definition 6.2.3. *$O = [s_1, s_2, \cdot\cdot, s_m]$ is a sub pattern of the pattern $O' = [s'_1, s'_2, \cdot\cdot, s'_n]$, denoted as $O \sqsubseteq O'$, if $m <= n$ and there exists a list of increasing integers $i_1, i_2, \cdot\cdot, i_m$ such that $1 \leq i_1 < i_2 < ... < i_m \leq n$, and $s_k = s'_{ik}$ for $k = 1, 2, \cdot\cdot m$. O is called the "sub pattern" while O' is denoted as the "super pattern".*

Let DS be a database of sensor lists; each list in a DS has an associated time called GT (Global Time); GT is the time that guarantees the correct order of sensors in the list (i.e., no sensor event that has a timestamp lower than that of the last sensor in the list will be received later). GT plays an important role in extracting data from WSNs. Table 6.2 shows an example for a database of five lists.

Table 6.1: Database of Lists (DS)

GT	***LIST***
10	$[(s_1, 2), (s_2, 3), (s_3, 5)]$
14	$[(s_3, 11), (s_2, 12), (s_4, 13)]$
20	$[(s_1, 15), (s_2, 17), (s_3, 18), (s_4, 19)]$
28	$[(s_1, 21), (s_4, 22), (s_2, 25), (s_3, 27)]$
35	$[(s_4, 29), (s_2, 32), (s_3, 33)]$

Definition 6.2.4. *Given a database DS of lists, let O be a chronological pattern; the support of the chronological pattern O in DS is defined by the number of lists in DS such that O is a sub pattern of the lists' chronological patterns.*

$$\textbf{\textit{Support(O)}} = |\{L \in DS | O \sqsubseteq O(L)\}|$$

Given database (DS) of lists and a minimum support (*min_sup*), the problem of mining chronological patterns basically comes down to finding all the chronological patterns whose support exceeds the given minimum support (these patterns are called frequent chronological patterns).

The first step in mining chronological patterns involves collecting data that describes the behavior of sensors (i.e., preparing the database (DS)); this first step is the focus of next section.

6.3 Data Preparation

In this section, we illustrate the process of preparing data needed for mining chronological patterns. The fact that Wireless Sensor Networks (WSNs) have limited resources necessitates the creation of a critical, efficient procedure to prepare the data. Most sensor networks use the publish/subscribe paradigm to deliver sensor readings [103, 104]. In this paradigm, the user injects a particular interest into the network via a well-equipped device (the Sink) that diffuses that interest to all sensors in the network. Each sensor maintains the received interest in a special table; upon detecting an event, a sensor checks its interest table for any interest that matches the detected event. If so, this event will be stamped with a current timestamp, and will be sent to the application. The application interprets the received events and delivers useful information to the user; however, it may happen that the application receives the events out of order, thus entailing the need for an ordering mechanism to be invoked before the events are delivered to the application. Ordering can be done by either initiating a particular ordering algorithm, like those in-

troduced in [38, 39], or by simply waiting a predefined length of time to confirm that all the events issued before the received events have arrived.

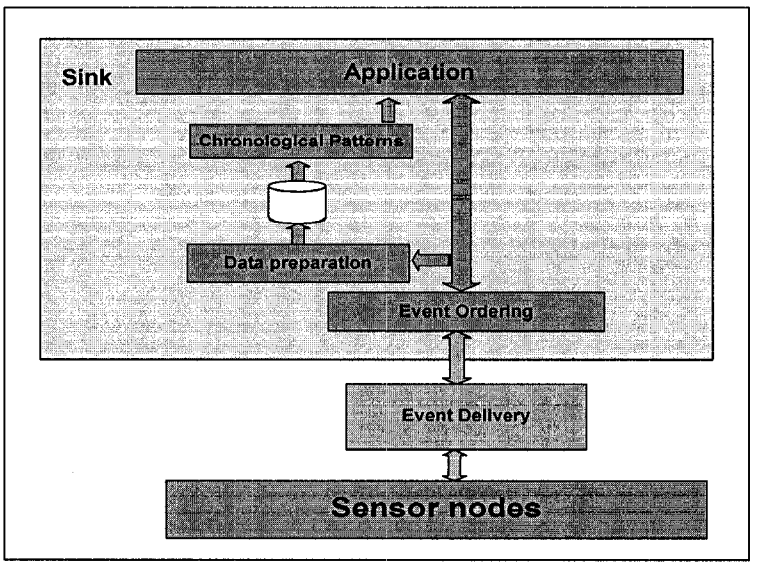


Figure 6.1: The Data Preparation Process.

Before delving further in this discussion, it must be emphasized that the data needed in mining chronological patterns is data that describes the behavior of sensors and differs from the actual data that contain sensor readings (i.e., meta-data about sensor behaviors). Although our proposed data preparation mechanism can be used with any data delivery method, we will assume a publish/subscribe paradigm (described above).

Figure 6.1 shows the procedure that we have adopted to prepare behavioral data needed in the mining process. The process starts by receiving events from the sensor network (events that carry the sensor readings). The event ordering layer will use its mechanism to ensure that the exact order of events before delivering them to the application. Also, this layer is responsible for accumulating the received events and delivering them to the application as a whole meaningful unit (in most scenarios, it is useful to deliver more than one event to the application at the same time in order to combine events from different sources and interpret useful information about the environment).

As an example, assume that the event ordering layer uses the delay mechanism (i.e., wait a predetermined amount of time after the last received event), and that the delay time is 5 units. Assume that at time 10 we have the list $[(s_1, 2), (s_2, 3), (s_6, 10)]$. Note that we have discarded the values of the events, as we had mentioned before, we are aiming to extract data that describes the behavior of sensors in detecting events, in spite of their readings. The event ordering layer waits until time 15 (i.e., the event layer decides that at time 10 we have a meaningful list of events) before sending these events (with their readings) to the application. Time 15 is called the Global Time (GT) with respect to the list $[(s_1, 2), (s_2, 3), (s_6, 10)]$; thus, at global time 15, the event layer delivers the list (along with the readings of the sensors) to the application, and at the same time stores this list (in an abstract manner without the sensor readings) in a particular database (DS) in order to use it in the mining process. The process continues until the end of the historical period defined by the application.

6.4 The Chronological Tree Structure Model

In the pervious section, we have introduced the data preparation mechanism that will prepare behavioral data, needed in mining the chronological patterns, from the received events and in cooperation with the event ordering mechanism employed by the network. The next step is to store the prepared data and apply a mining algorithm that can generate the frequent chronological patterns (i.e., the chronological patterns in the database whose frequencies exceed the given minimum support).

Real world environments are active domains that generate a large number of events in a short period [1]; this in turn results in vast volumes of meta-data being accumulated in a short time at the Sink node. This being the case, the need for an efficient data structure emerges: one that can compress data and make it accessible, thus accelerating the mining process. In this section, we present the Chronological Tree (CT) structure model, our proposed data structure for storing and compressing behavioral data (meta-data). CT

has possesses common concepts with the Positional Lexicographic Tree (PLT) structure, which was proposed in chapter four, however, many concepts have been modified to reflect the sequence nature of the patterns. In what will follow, we explain the main concepts of CT and its construction process, and then present the mining algorithm that generates the frequent chronological patterns from the CT efficiently.

6.4.1 The Chronological Tree Construction Process

The Chronological Tree's 'mechanism' uses the lexicographic distance between identifiers of sensors that appear in the same chronological pattern to represent the data in a more compressed format. The major difference between the lexicographic distance in the context of CT and its equivalent in PLT is that in CT, negative numbers are allowed to appear as values assigned to the lexicographic distance between sensor identifiers; this is attributed to the aspect of ordering of sensors, as opposed to PLT, where sensors are sorted according to the decreasing order of their frequencies (before construction of positional vectors commences). It should be recalled here that sorting cannot be performed on chronological patterns; it affects the order in which sensors in the pattern have detected the events.

We will start with some definitions, and then present the construction process of the Chronological Tree in more details.

Let S be the set of sensors in a particular WSN, a lexicographic order is assumed to exist among the sensor identifiers (i.e., s_1 occurs lexicographically before s_2).

Definition 6.4.1. *Rank (s) is a function that maps each sensor ($s \in S$) to a unique integer in such a way that the lexicographic order is maintained.*

The rank function is used to re-assign integer identifiers to sensors to induce regularity to data (i.e., no big gaps between sensor identifiers).

Definition 6.4.2. *Given a chronological pattern $O = [s_1, s_2, \cdot\cdot, s_n]$, Order(s) is a function that maps each sensor in the pattern to an integer value that represents its rank's*

lexicographical distance from the rank of the left neighbor in the pattern. This value is positive if the sensor's rank is lexicographically greater than the rank of the left neighbor, and negative if less.

The order function captures the lexicographical distance between sensor identifiers and is defined to be the difference between the sensor's rank and the rank of the left neighbor in the list's chronological pattern. Recall that the chronological pattern is constructed by removing the time stamp associated with each sensor in the list. As an example, let O = $[s_3, s_4, s_1]$ be a chronological pattern, and let the sensors' ranks in this pattern be 3, 4, and 1 respectively, then Order(s_3) = 3, Order(s_4) = 1, and Order(s_1) = -3 (i.e., s_1 is lexicographically 3 units lower than s_4). Note that the order of the first sensor in the list is the same as its rank (as if there is no left neighbor).

Definition 6.4.3. *Let* $O = [s_1, s_2, \cdot\cdot, s_n]$ *be a chronological pattern of length n,* $V(O) = [Order(s_1), Order(s_2), \cdot\cdot, Order(s_n)]$ *is defined to be the chronological vector of the pattern O.*

Lemma 6.4.1. *Let* $V(O) = [Order(s_1), Order(s_2), \cdot\cdot, Order(s_n)]$ *be the chronological vector of the pattern O, for each* $s_i \in O$. *It then holds that*

$$Rank(s_i) = \sum_{j=1}^{i} Order(s_j)$$

After defining the major concepts, we will illustrate the process of converting the lists in the behavioral database into a more compressed structure using the Chronological Tree (CT). As in the case of the positional lexicographic tree, the physical view of the CT consists of several partitions in which each partition stores the order vectors that share a common sensor in their patterns. We use the notation CT[x] to refer to the CT partition that stores the vectors that share the sensor with rank x. The key point of the CT construction process is to convert each list's chronological pattern in the database, after removing the infrequent sensors from the pattern, to an order vector, and then store the

vector in the partition that is indexed with an integer value that represents the maximum rank among the sensor's ranks in the corresponding pattern of the vector. The position of the sensor with the maximum rank in the chronological pattern differs from the position of its equivalent in the association patterns (in the former, it may be any sensor in the pattern, in contrast to the case in the association pattern, where this sensor is the last sensor in the pattern). In what will follow, we will illustrate this process using a running example.

Table 6.2: Database of Lists (DS)

GT	***LIST***
10	$[(s_1, 2), (s_2, 3), (s_3, 5)]$
22	$[(s_3, 12), (s_2, 15), (s_4, 17)]$
32	$[(s_1, 23), (s_2, 25), (s_3, 26), (s_4, 27)]$
45	$[(s_1, 33), (s_4, 35), (s_2, 37), (s_3, 40)]$
53	$[(s_4, 46), (s_2, 45), (s_3, 48)]$

Consider the database of lists presented in Table 6.2, and assume that the minimum support is equal to two. The construction process starts by determining the set of frequent sensors (i.e., the sensors that will participate in formulating longer frequent chronological patterns); these sensors are $\{(s_1, 3), (s_2, 5), (s_3, 5), (s_4, 4)\}$. Ranks are then assigned to these sensors as follows: 1, 2, 3, and 4, respectively, note that no sorting is performed for these sensors. Afterwards, another scan of the database is conducted, and for each list's chronological pattern presented in the database, the set of infrequent sensors is filtered out and the corresponding chronological vector of remaining sensors is generated. For instance, the first list in the database in Table 6.2 exhibits the chronological pattern $[s_1, s_2, s_3]$, which is associated with a chronological vector equivalent to [1, 1, 1]. s_3 is the sensor with maximum rank in the corresponding pattern of this vector, and is therefore stored in the CT partition that is indexed with 3 (i.e., CT[3] is the chronological tree partition that is indexed with 3); recall that the chronological pattern of a list is the list of sensors appearing in the pattern without their time stamps. For

CT[4]

Order(1)	Order(2)	Order(3)	Order(4)	Support	Com
4	-2	1		1	21
1	3	-2	1	1	15
3	-1	2		1	14
1	1	1	1	1	4

CT[3]

Order(1)	Order(2)	Order(3)	Support	Com
1	1	1	1	3

Ranks are s_1 =1, s_2 =2, s_3 =3, s_4 =4.

Figure 6.2: The Chronological Tree (Physical View).

the remaining lists, they have the following vector representations: [3, -1, 2], [1, 1, 1, 1], [1, 3, -2, 1], and [4, -2, 1], respectively. These vectors have sensor s_4 as the sensor with maximum rank in their corresponding chronological patterns, and are stored in the CT's partition that is indexed with 4 (i.e., CT[4]). Figure 6.2 shows the physical view of the chronological tree for the database presented in Table 6.2.

Each entry in a CT partition-one for each vector-consists of three fields: (i) vector filed that stores the order values of the vector; (ii) frequency field that stores the frequency of the vector in the database; (iii) comparison value that captures the distance of the vector from the origin, and has the same structure and usage like the one used in the position lexicographic tree.

At this point, we can verify our choices regarding the structure of the chronological tree: first, we decided on the chronological vectors to represent the lists because of their compressed structure and their ability to easily identify the sensors presented in these vectors by using Lemma 6.4.1. The compressed structure and the partitioning property of the *CT* make it the best choice for distributed systems. Second, we chose to keep CT's partitions in a sorted format to speed the process of inserting and locating vectors; the sorted format is maintained using two types of ordering: The first order is based on the comparison value that is associated with each vector. The comparison value captures

the distance of the vector from the origin and defined as follows:

$$V(L(O)).Com = \sum_{j=1}^{i} Order(s_j)^2 \tag{6.1}$$

The comparison value is used in the process of inserting or locating a vector in CT and defines a partial ordering among the vectors. For example, given two vectors with comparison values com_1 and com_2, $vector_1$ will appear before $vector_2$ if $com_1 > com_2$. The second order is the lexicographic order, which is used when the comparisons values are equal (i.e., when $com_1 = com_2$, $vector_1$ and $vector_2$ will be compared lexicographically). The comparison value is also used as an index for locating the vectors.

To summarize, mining CT occurs in two steps: the first step realizes the construction of the chronological tree (CT), which is a compact representation of the behavioral data. Each vector in this CT will be associated with a frequency value that reflects its frequency in the database, and a comparison value that is used as an index for sorting and locating the vectors in the database; the second step involves mining the CT to generate the chronological patterns that meet the minimum support, which is the focus of the next section. Algorithm 8 shows a formal description for CT construction process.

Algorithm 8 The CT Construction Process

CT_Construction(DS, Min_sup)
Fs = Frequent sensors.
Assign ranks to Fs element.
For each $L \in DS$
    Let $L' = [s_1, \cdot\cdot, s_k]$the frequent sensors in L
    $V(L') = [Order(s_1), \cdot\cdot, Order(s_k)]$
    $V_{L'}.Com = \sum_{i=1}^{k} Order(s_i)^2$
    Index = Max rank in L'.
    Insert V(L') to CT[Index]

6.4.2 The Chronological Tree Mining Process

In this section, we present the mining process that generates the frequent chronological patterns from the Chronological Tree (CT). The proposed mining process follows the pattern growth approach [24], and considers each CT partition separately (in decreasing order of partition indices); the goal is to group all the lists in which a particular pattern participates in one structure (the conditional structure of the pattern [24]), and then generate all the frequent chronological patterns in which this pattern participates, without considering any other lists. We use the notation *CT_P* to refer to the conditional structure of the pattern P (i.e., all the lists that support the chronological pattern P are presented in *CT_P*).

As in the Positional Lexicographic Tree (PLT), the CT mining process is a recursive process that employs the concept of conditional structure. Each CT partition is considered separately (in decreasing order of partition indices), and undergoes several recursive steps until all the frequent chronological patterns it contains are generated-Recall that new partitions may appear in the CT due to the update process that we will discuss later. Initially, the frequent sensors presented in the CT are considered as patterns of length one, and are deemed the seeds to build longer frequent patterns. Each pattern's conditional structure is a new CT structure that stores the lists that support it, and is constructed from its conditional vectors. In what will follow, we will start by giving some definitions pertaining to the CT mining process, and then describe the process in more details using a running example.

Definition 6.4.4. *The conditional vectors of sensor s_i are the chronological vectors' representations of the suffixes extracted from the patterns' representation of the vectors in CT [Rank(s_i)] from the sensor identifier that follows the sensor s_i to the end of the pattern.*

Definition 6.4.5. *Let CT_P be the conditional CT structure of the pattern P. Let s_i be one of the sensors in this conditional structure. The conditional structure of the pattern*

$P + s_i$ (i.e., $CT\_(P + s_i)$) is the CT structure built using the patterns constructed from the conditional vectors of s_i in $CT\_P[Rank(s_i)]$.

We begin with the CT partition that is indexed with the maximum rank, assume the maximum rank is r, then, from CT [r] (i.e., the chronological tree partition that is indexed with rank r) we produce the conditional vectors of the sensor with rank r (see definition 6.4.4). Back to the example in Figure 6.2, starting from CT[4], we extract the conditional vector [2, 1] that refers to the pattern $[s_2, s_3]$ from the vector [4, -2, 1]. Recall that the vector [4, -2, 1] refers to the pattern $[s_4, s_2, s_3]$, which will produce the suffix $[s_2, s_3]$ whose chronological vector representation is [2, 1]. Also, we extract the conditional vector [2, 1] from the vector [1, 3, -2, 1]. Note that nothing can be extracted from the vectors [1, 1, 1, 1] and [3, -1, 2], since there are no suffixes that can be extracted from their patterns. Next, we construct the conditional chronological tree of s_4 (CT_s_4), which has the same structure as the original CT (see definition 6.4.5). The main difference between the original CT and the conditional structure is that the conditional structure is built from the patterns that have been generated from the patterns' representation of the conditional vectors. Figure 6.3 shows CT_s_4, where the frequent sensors that appear in this structure are $\{(s_2, 2), (s_3, 2)\}$, the number associated with each sensor is the support count for the sensor in this conditional structure. Recall that new ranks have been assigned to the sensors in the new conditional structure.

$CT\_s_4[2]$

Ord(1)	Ord(2)	Support	Com
1	1	2	2

Frequent sensors are $\{(s_2, 2), (s_3, 2)\}$
Ranks are s_2 =1, s_3=2.

Figure 6.3: CT_s_4

Before going further in the mining process, we have to mention that the CT in which the conditional vectors are extracted must be updated. These updates should occur at any stage in the mining process when conditional vectors are extracted. In the

update process, two cases should be distinguished based on the position of the sensor identifier-which we are in process of extracting its conditional vectors-in the pattern's representation of the vector from which the suffix is going to be extracted (we refer to this sensor by split value); these cases include:

1. If the split value is the first value in the pattern's representation of the vector, the *conditional vector* is added to the database partition that is indexed with the rank of the sensor with the maximum rank presented in the pattern representation of the conditional vector extracted from this vector.

2. If the split value is somewhere else in the vector, other than the first value, the *original vector* (not the extracted) is inserted into the CT partition with index equal to the sensor's rank that is immediately less than the rank of the split value; if there is no such rank, this vector is just discarded.

As an example of the first case in the update process, the conditional vector [2, 1] (that refers to the pattern $[s_2, s_3]$) that has been extracted from the vector [4, -2, 1] (when s_4's conditional vector was extracted from CT,) should be added to CT[3] (i.e., 3 is the maximum rank among the sensors' ranks in the pattern $[s_2, s_3]$). Vectors [1, 3, -2, 1], [1, 1, 1, 1] and [3, -1, 2] are examples of the second case; these vectors are added to CT[3], as the original patterns for these vectors are $[s_1, s_2, s_3, s_4]$, $[s_1, s_4, s_2, s_3]$, and $[s_3, s_2, s_4]$, respectively. 3 is the rank in these patterns, which is immediately less than 4. Figure 6.4 shows the updated CT after extracting s_4's conditional vectors.

The mining process proceeds from CT_s_4 (i.e., (s_4)'s conditional structure), Figure 6.3; it starts from the sensor with the maximum rank, s_3, which has rank equal to 2. At this stage, the first frequent chronological pattern $[s_4, s_3]$ is produced with a support count equal 2, and the conditional vectors of s_3 are extracted from $CT\_s_4[2]$. At the same time, the CT_s_4 is updated using the updating procedure described above. Figure 6.5 shows CT_s_4 after the update process.

CT[3]

Order(1)	Order(2)	Order(3)	Order(4)	Support	Com
1	3	-2	1	1	15
3	-1	2		1	14
2	1			1	5
1	1	1	1	1	4
1	1	1		1	3

Ranks are s_1 =1, s_2 =2, s_3 = 3.

Figure 6.4: The Updated CT after Extracting s_4's Conditional Structure.

CT_s_4[1]

Ord(1)	Ord(2)	Support	Com
1	1	2	2

Frequent sensors are $\{(s_2, 2), (s_3, 2)\}$
Ranks are s_2 =1, s_3 =2.

Figure 6.5: The Updated CT_s_4 after Extracting s_3's Conditional Vectors

Since there is no conditional structure for the pattern $[s_4, s_3]$ in $CT\_s_4[2]$ (as no suffixes can be extracted from the pattern representation of the vector [1, 1]), the process continues by considering the next frequent sensor in CT_s_4, s_2, that has rank equal to 1. At this stage, the second frequent pattern is generated, $[s_4, s_2]$, with a support count equal to 2, and the conditional structure of $[s_4, s_2]$ is built (i.e., $CT\_s_4s_2$). Sine 1 is the last rank in CT_s_4, no updating occurs to this structure when the conditional vectors of s_2 are extracted. Figure 6.6 shows the conditional structure of the pattern $[s_4, s_2]$.

The mining process then proceeds from $CT\_s_4s_2$ (Figure 6.6), and starts with the sensor with the maximum rank, s_3 (equal to 1), and generates the pattern $[s_4, s_2, s_3]$ with

$CT\_s_4\_s_2$[1]

Ord(1)	Support	Com
1	2	1

Frequent Sensors are $\{(s_3, 2)\}$
Ranks are s_3 =1.

Figure 6.6: $CT\_s_4s_2$

a support count of 2. Sine there is no conditional structure for the pattern $[s_4, s_2, s_3]$ in $CT\_s_4s_2[1]$, and no more ranks to be considered in $CT\_s_4s_2$ or in CT_s_4, the process returns to consider the sensor that has the rank immediately below (s_4)'s rank in the original CT (Figure 6.4), which is s_3.

The mining continues by extracting the conditional vectors of the sensor s_3 in the CT (Figure 6.4): [2, 2] with frequency 1, and [4] with frequency 1; they refer to the patterns $[s_2, s_4]$ and $[s_4]$, respectively. At the same time the CT is updated (Figures 6.8). The conditional structure of CT_s_3 is then built from the extracted vectors (Figure 6.7). s_4 (rank equal to one) is the only sensor frequent in CT_s_3, so we generate the patterns $[s_3, s_4]$ with frequency 2. Since there no conditional vectors for this pattern, and no more sensors to consider in CT_s_3, the process returns recursively to consider the sensor in CT (Figure 6.8) with rank immediately below 3 (which is 2) that refers to sensor s_2.

$CT\_s_3[1]$

Order(1)	Support	Com
1	2	1

Frequent sensors are $\{(s_4, 1)\}$
Ranks are s_4 =1.

Figure 6.7: The CT_s_3 Chronological Tree

$CT[2]$

Order(1)	Order(2)	Order(3)	Order(4)	Support	Com
1	3	-2	1	1	15
2	2			1	8
2	1			1	5
1	1	1	1	1	4
1	1	1		1	3

Ranks are s_1 =1, s_2 =2, s_3 = 3, s_4 = 4.

Figure 6.8: The Updated CT after Extracting s_3's Conditional Structure.

The process continues from the original CT (Figure 6.8) by extracting the conditional vectors of s_2: [3] with frequency of 3, [4] with frequency of 1, and [3, 1] with frequency 1

$CT\_s_2[2]$

Order(1)	Order(2)	Support	Com
1	1	1	2
2		1	1

$CT\_s_2[1]$

Order(1)	Support	Com
1	3	1

Frequent sensors are $\{(s_3, 4), (s_4, 2)\}$
Ranks are s_3 =1, s_4 =2.

Figure 6.9: The CT_s_2 Chronological Tree

(that refer to the patterns $[s_3]$, $[s_4]$, and $[s_3, s_4]$, respectively). From these patterns, we build CT_s_2 (Figure 6.9), while at the same time updating the original CT (Figure 6.10).

$CT[1]$

Order(1)	Order(2)	Order(3)	Order(4)	Support	Com
1	3	-2	1	1	15
1	1	1	1	1	4
1	1	1		1	3

Ranks are s_1 =1, s_2 =2, s_3 = 3, s_4 =4.

Figure 6.10: The Updated CT after Extracting s_2's Conditional CT.

After building the CT_s_2 structure, the process proceeds by considering the maximum rank among this structure's sensor ranks (2) that refers to sensor s_4, and produces the patterns $[s_2, s_4]$ with frequency 2. At the same time, CT_s_2 is updated (Figure 6.11). Since there are no conditional vectors for sensor s_4 that can extracted from CT_s_2, the process proceeds by considering the next rank in the updated CT_s_2 (Figure 6.11), which is 1 (i.e., sensor s_3), and generates the patterns $[s_2, s_3]$ with frequency 4. Recall that there is no update process for CT_s_2, since 1 is the last rank presented in this structure. Since there is no conditional structure for $[s_2, s_3]$, as there is no frequent sensors in its extracted conditional vectors, and no more ranks to consider in CT_s_2, the mining process returns recursively to consider the next rank in the CT (Figure 6.10): 1(refers to sensor s_1).

The mining process then proceeds to consider the last sensor, s_1, whose rank is 1.

$CT\_s_2[1]$

Order(1)	Order(2)	Support	Com
1	1	1	2
1		3	1

Frequent sensors are $\{(s_3, 4), (s_4, 1)\}$
Ranks are s_3 =1, s_4 =2.

Figure 6.11: The Updated CT_s_2 Chronological Tree

Recall that no update will occur for this partition after extracting s_1's conditional vectors, as it is the last partition in the chronological tree. The conditional vectors for s_1: [4, -2, 1], [2, 1, 1], [2, 1] refer to the patterns $[s_4, s_2, s_3]$, $[s_2, s_3, s_4]$, and $[s_2, s_3]$, respectively, all of which have frequency equal to one. Then, from these patterns the conditional structure of s_1 (i.e., CT_s_1) is built, Figure 6.12.

The mining process continues by considering the partition indexed with maximum rank in CT_s_1, which is 3, and refers to s_4, (Figure 6.12). The frequent pattern $[s_1, s_4]$ with support count equal to 2 is generated. The mining procedure then extracts the conditional vector of s_4 in and updates CT_s_1 (Figure 6.13). Since there are no frequent sensors in s_4's conditional vectors, no conditional structure can be built for s_4.

$CT\_s_1[3]$

Order(1)	Order(2)	Order(3)	Support	Com
3	-2	1	1	14
1	1	1	1	3

$CT\_s_1[2]$

Order(1)	Order(2)	Support	Com
1	1	1	2

Frequent sensors are $\{(s_2, 3), (s_3, 3), (s_4, 2)\}$
Ranks are s_2 =1, s_3 = 2, s_4 =3.

Figure 6.12: The CT_s_1

Next, the mining process considers the partition in CT_s_1, which has index equal to 2 (Figure 6.13). The frequent pattern $[s_1, s_3]$ is then generated with a support count 3 and the conditional vectors of s_3 are extracted. At the same time, CT_s_1 is updated.

$CT\_s_1[2]$

Order(1)	Order(2)	Order(2)	Support	Com
1	1	1	1	3
1	1		2	2

Frequent sensors are $\{(s_2, 3), (s_3, 3), (s_4, 1)\}$
Ranks are s_2 =1, s_3 = 2, s_4 =3.

Figure 6.13: The Updated CT_s_1 after Extracting s_4

Figure 6.14 shows the updated CT_s_1. Since there are no frequent sensors in s_3's conditional vectors, the process returns to consider the last partition in CT_s_1, which is indexed with 1 (Figure 6.14)

$CT\_s_1[1]$

Order(1)	Order(2)	Order(2)	Support	Com
1	1	1	1	3
1	1		2	2

Frequent sensors are $\{(s_2, 3), (s_3, 3), (s_4, 1)\}$
Ranks are s_2 =1, s_3 = 2, s_4 =3.

Figure 6.14: The Updated CT_s_1 after Extracting s_3

From $CT\_s_1[1]$, the pattern $[s_1, s_2]$ is generated with support count 3, and the conditional structure of this pattern is built (Figure 6.15) from the patterns $[s_3, s_4]$ and $[s_3]$, which have frequencies 1 and 2, respectively. Recall that no update is performed for CT_s_1 as it has been the last partition that the mining process was 'working on' in this structure.

$CT\_s_1s_2[1]$

Order(1)	Support	Com
1	3	1

Frequent sensors are $\{(s_3, 3)\}$
Ranks are s_3 =1.

Figure 6.15: The $CT\_s_1s_2$

The process continues from $CT\_s_1s_2[1]$ and produces the pattern $[s_1, s_2, s_3]$ with frequency 3. Since this is the only partition in $CT\_s_1s_2$, and there are no more partitions to

Table 6.3: The Chronological Patterns

Chronological Pattern	Support
s_4	4
s_4s_3	2
s_4s_2	2
s_3	5
s_3s_4	2
s_2	5
s_2s_4	2
s_2s_3	4
s_1	3
s_1s_4	2
s_1s_3	3
s_1s_2	3
$s_1s_2s_3$	3

consider in CT_s_1 or in the original CT, the mining process stops at this point. Table 6.3 shows all the frequent chronological patterns presented in the database in Table 6.2. Algorithms 9 and 10 show formal descriptions for CT mining and update process.

Algorithm 9 The Mining Process

```
Main()
  Index = Maximum rank in CT
  For i = Index downto 1
      CT_Mining(CT, i, φ)
  Endfor

CT_Mining(CT, i , P)
  CV_i = the conditional vectors of i
  Update(CT, i)
  CE = the set of patterns constructed from CV_i
  P=P + "i"
  Output(P)
  CT_P = CT_Construction(CE, Min_Sup)
  Cindex = Maximum rank in CT_P
  For j = Cindex downto 1
       CT_Mining(CT_P, j, P)
  End{for}
```

Algorithm 10 The Update Process

```
Update(CT, r)
For each Vector v in CT[r]
    Let o_n be the split value
    Let v' be the conditional vector extracted from v
    If o_n is the first value in v
        Max = Maximum rank in v''s pattern representation.
        Insert(v',CT[Max])
    Else
        Max = Maximum rank in v's pattern representation that is less than r
        Insert(v,CT[Max])
    End If
End For
```

6.5 Performance Evaluation

In order to document the performance of the Chronological Tree, we have evaluated its performance using different data sets generated at different density factors. Density factor defines an ordering correlation between a set of sensor nodes. We believe that it may be better to evaluate the performance of the chronological tree as compared with other similar schemes, however, most algorithms proposed in the literature for these kind of patterns assume that each element in the pattern is a set of objects, in contrast to the case of chronological patterns, which assumes each element in the pattern is one object (i.e., sensor node).

Coding for the Chronological Tree (CT) has been implemented using C++ compiled with g++ version 3.4.4. For the data sets, we have used a simulator that has been built using Matlab version 7.4.0.287 [101] to generate synthetic data. The simulator generates the data using different parameters, including the number of nodes, historical period, and a density factor. The historical period defines the period of events generation, which we have assumed to be every 60 seconds. The density factor defines the degree of correlation between the nodes; we have chosen a density factor of .015 and .020 achievable at a minimum support of 50%. The average length of an epoch is assumed to fall between 0.25% and 0.3% of number of nodes. Different data sets have been generated using 500 and 1000 nodes, and a historical period of ten days has been fixed for all sets. The

measurements we are interested in reporting about the chronological tree are CPU time, memory usage, and number of patterns. The CPU time and memory usage measurements comprise the time and memory consumed for the CT construction and mining process. These measurements have been collected using the (memusage) and (time) commands available in Linux operating system. Table 6.4 summarizes the characteristics of the data sets used in the performance evaluation of the chronological tree.

Data	**# nodes**	His_period (day)	**Density Factor**
S1000H10D-.015	1000	10	.015
S1000H10D-.020	1000	10	.020
S500H10D-.015	500	10	.015
S500H10D-.020	500	10	.020

Table 6.4: Data Set Characteristics

Figures 6.16 to 6.19 show the CPU time and the memory usage for the data sets presented in Table 1.4 with minimum support values ranging from 50% to 90%. As can be inferred from the figures, the chronological tree can be a good representational structure for the data and a useful tool for generating the chronological patterns using reasonable amount of CPU time and memory usage. For all the data sets, the execution time ranges from 0.5 to 7 seconds and memory consumption between 15 to 600 MB. Also, it is evident from the figures how both time and memory usage increase rapidly at support value 50% for the different density factors; this is due to the combinatorial characteristics of the chronological patterns, where the number of patterns increases dramatically as the density factor increases. In contrast, CPU execution time and memory usage continue close to each other during the period that sees the support values increase; this is due to the reduction in the number of frequent chronological patterns at higher support values.

Our choice for density factors is based on the fact that correlation will exist between sensors that are close together. However, these density factors will be more related to the coverage properties of sensor nodes. Figure 6.20 shows an analytical result for the expected number of patterns versus different density factors. In this analytical figure, we

have assumed that each data set has a certain number of nodes (i.e., the density factor) that are correlated and appear together in every epoch. To make the figures more readable, we have used the logogrammatic number of patterns to the base 10 to draw the figure. As we can see in the figure, the number of patterns increases dramatically when the density factor increases, which explains the sudden jumps in CPU time and memory usage in the pervious figures. In real scenarios, however, the density will be distributed among disjoint groups of sensors, and it will be feasible to generate the frequent patterns efficiently due to the partitioning mechanism of the Chronological Tree.

Although the update process for the chronological tree performed during the mining process is more complicated than its counterpart in the PLT, the factors that enable PLT to outperform the FP-Growth are still applicable for the chronological tree

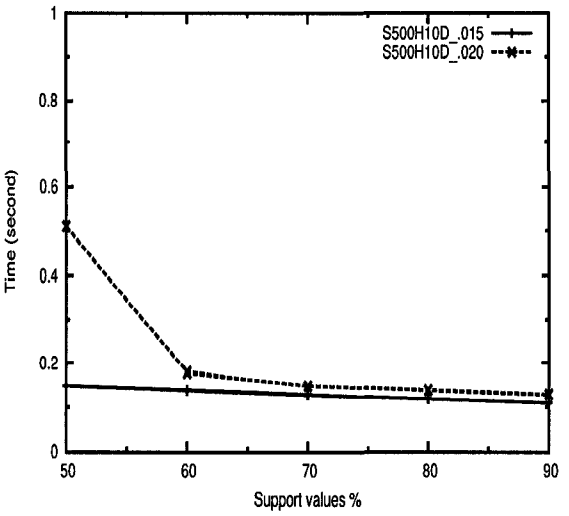


Figure 6.16: CT CPU Time for 500 Nodes

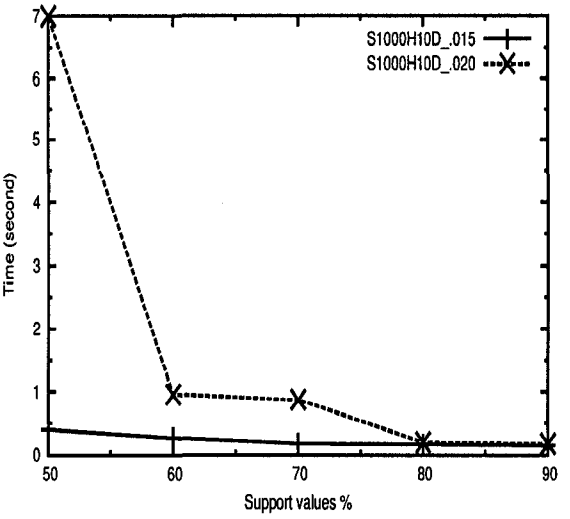


Figure 6.17: CT CPU Time for 1000 Nodes

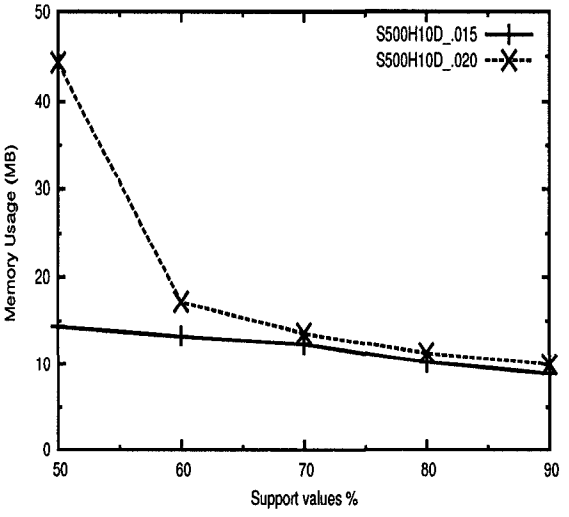


Figure 6.18: CT Memory Usage for 500 Nodes

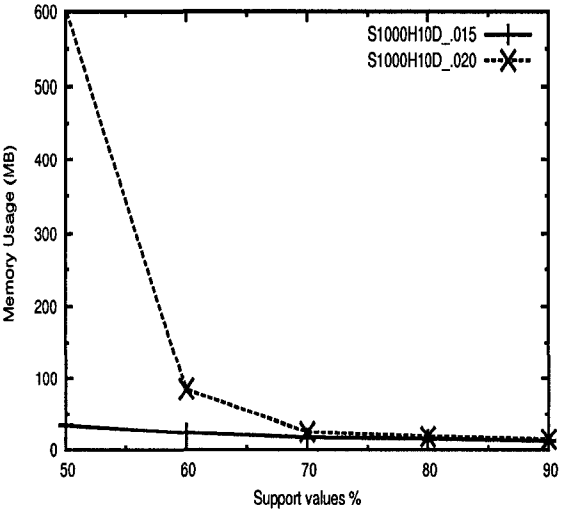


Figure 6.19: CT Memory Usage for 1000 Nodes

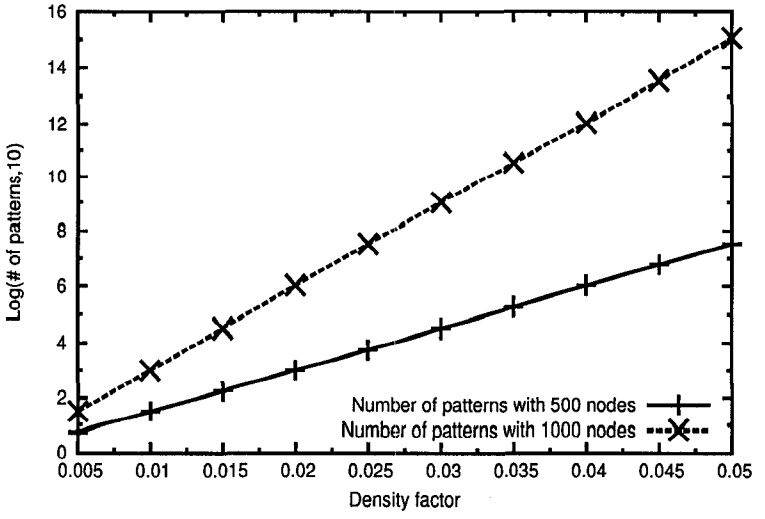


Figure 6.20: Number of Patterns Versus density Factors with 500 and 1000 Nodes

6.6 Summary

Chronological Patterns is an attempt to capture the order-temporal correlations between sensor nodes in WSNs. A direct application of Chronological Patterns is reducing the undesired cons of wireless communication. In this chapter, we have given a formal definition of Chronological Patterns in terms of sensor nodes, and presented a data preparation mechanism to collect the data needed to generate the patterns. We have also proposed a data representation structure, which we have referred to as Chronological Tree (CT), to maintain the data and present it in a way that leads to an efficient mining of patterns. To evaluate the performance of the CT, several experiments have been conducted using different settings and density factors. Results have confirmed the feasibility of implementing the CT, to certain degree of density factors, with reasonable amount of resources.

Chapter 7

Conclusion and Future Work

7.1 Final Remark

The main focus of this research has been to propose a comprehensive framework for the Knowledge Discovery process in Wireless Sensor Networks (WSNs). The goal of this process is to extract behavioral patterns regarding the sensor nodes during their operation. As far as we are concerned, this is the first study that generated patterns regarding sensor nodes, in contrast to the other works that had focused on generating patterns regarding the area under monitoring. Three types of behavioral patterns have been proposed: Sensor Association Rules, Coverage-based Rules and Sensor Chronological Patterns. For each of these pattern, we have successfully defined the proper steps in the Knowledge Discovery process to answer the questions: "How are the behavioral patterns defined in terms of sensor terminologies?"; "How is the metadata needed in the mining step extracted?"; and "How are patterns generated from the metadata?" The main' duty' of behavioral patterns is to enhance the performance of WSNs and their Quality of Services (QoSs) through participating in the resource management process and compensating for the undesired affects of wireless communication. The design choices in the proposed framework have been selected to take into consideration the properties of WSNs (e.g., the limited resources, stream nature, etc...).

Sensor Association Rules captures the temporal correlations, of detecting events, between sensor nodes. Three different mechanisms have been proposed for preparing the data needed for generating sensor rules: Direct Reporting, Distributed Extraction and In-network Reduction. Based on the reported results, In-network Reduction, through using the buffer strategy and the deference operation, has shown promising reduction in the number of messages and average energy consumption. To meet the requirements of large volumes of data, a storage structure called Positional Lexicographic Tree (PLT) has been proposed. PLT uses the lexicographic distance between sensor identifiers to compress data, and employs a partitioning mechanism to partition the data into several structures. For generating the frequent patterns, a mining algorithm has been proposed; it follows the pattern growth approach. The partition mechanism of PLT makes it a good choice for a distributed solution for mining Sensor Association Rules. Different experimental results have been introduced to compare the PLT structure with FP-Growth, a well known mining algorithm in this aspect. Results have shown the outperforming of the PLT structure in term of CPU time and memory usage.

Coverage-based Rules has been designed specifically for sensor networks that guarantee a k-coverage property for the area under monitoring. In contrast to Sensor Association Rules, Coverage-based Rules discovers the relation between a set of locations instead of between the sensor nodes in the network. The logic behind this technique is that sensor nodes covering the same location will share the same activity sets. Generating associations between all the sensor nodes will result in a redundant information to the Sink that already known by the coverage property of the network. Compared to Sensor Association Rules, Coverage-base Rules achieves a great reduction in messages and of average energy consumption.

Chronological Patterns focuses on discovering the temporal-ordered correlation between sensor nodes. Chronological Patterns exudes a similar structure to Sensor Association Patterns, however, its data preparation, storage structure and mining algorithm have been designed to reflect the sequence and combinatorial natures of these patterns.

Several experimental results have been employed to evaluate the performance of the CT using different data sets, prepared for this purpose. Results have shown the effectiveness of generating these patterns using a reasonable amount of resources.

Based on the performance results presented, we believe that we have successfully extended the Knowledge Discovery domains to include WSNs, and have provided the possibility of a real implementation of the proposed behavioral patterns to enhance the performance of WSNs.

7.2 Summary of Thesis Contribution

The main objective of the thesis was to define the major steps in the Knowledge Discovery process for extracting behavioral patterns. Our contribution to the knowledge process in wireless sensor network has been achieved through re-defining the main steps in the process to take into consideration the limited resources of WSNs. The main contributions of the thesis are summarized as follows:

- Regarding the Knowledge Definition step, three behavioral patterns have been proposed:

  i *Sensor Association Rules.*

  ii *Coverage-based Rules.*

  iii *Chronological Patterns.*

- Regarding the data preparation step, the following schemes have been proposed

  - Three different data extraction mechanisms for preparing the data needed for generating Sensor Association Rules; these are:

    i *Direct Reporting.*

    ii *Distributed Extraction.*

    iii *In-network Data Reduction.*

  - A data preparation mechanism for preparing the data needed for generating the Chronological Patterns implemented on top of an event ordering algorithm.

- Regarding the data mining steps, the following structure has been proposed:

  i A *Positional Lexicographic Tree (PLT)* for mining Sensor Association Rules.

  ii A *Chronological Tree* for mining Chronological Patterns.

7.3 Future work

Several future research directions can be realized by extending the work presented in this thesis; these include:

1. Investigating the challenges of implementing the proposed framework in a Wireless Actor and Sensor Network (WASN) architecture. Actors are nodes with better processing capabilities, higher transmission powers and more energy. As reported in [83]," sensors in WASN gather information about the physical world, while actors take decisions and then perform appropriate actions upon the environment, which allows a user to effectively sense and act from a distance".

2. Employing a Data Reduction scheme to implement a data preparation mechanism for Coverage-based Rules. The envisioned mechanism will take advantage of the redundancy that exists between data.

3. Exploring the possibility of implementing the Positional Lexicographic Tree (PLT) and the Chronological Tree (CT) in a distributed fashion using multiple and mobile Sinks.

4. Studying the limitations of making the proposed schemes fault-tolerant. Adding fault tolerance requires re-defining the proposed patterns to include the concept of faulty nodes.

5. Simulating several scenarios to measure the effectiveness of the proposed patterns in enhancing the performance of wireless sensor networks.

6. Implementing the proposed framework in a real wireless sensor network.

7. Extending the proposed framework to include more knowledge-based concepts, such as prediction and classification.

Bibliography

[1] Zhao, F., and Guibas, L.J. Wireless Sensor Networks: An Information Processing Approach. Morgan Kaufmann publisher, 2002.

[2] Han, J., and Kamber, M. Data Mining: Concepts and Techniques, 2^{ed}. Morgan Kaufmann publisher, 2006.

[3] Witten, I.H, and Frank, E. Data Mining: Practical Machine Learning Tools and Techniques, 2^{ed}. Morgan Kaufmann publisher, 2005.

[4] Duda, R. O., Hart, P. E., and Stork, D. G. Pattern Classification, 2^{ed}. John Wileyand Sons, 2001.

[5] Fayyad, U., Piatetsky-Shapiro, G., and Smyth, P. From Data Mining to Knowledge Discovery: An Overview. In Advances in Knowledge Discovery and Data Mining, pp. 1-34, 1996.

[6] Fayyad, U., Piatetsky-Shapiro, G., and Smyth, P. From Data Mining to Knowledge Discovery in Databases. AI Magazine 17(3), pp. 37-54, 1996.

[7] Dunham, M. H. Data Mining: Introductory and Advanced Topics, 1st. Prentice Hall PTR, 2002.

[8] Boukerche, A., Pazzi, R. W., and Araujo, R. B. A fast and reliable protocol for wireless sensor networks in critical conditions monitoring applications. In Proceedings

of the 7th ACM international Symposium on Modeling, Analysis and Simulation of Wireless and Mobile Systems, pp. 157-164, Venice, Italy, 2004.

[9] Römer, K., Kasten, O., and Mattern, F. Middleware challenges for wireless sensor networks. SIGMOBILE Mob. Comput. Commun. 6(4), pp. 59-61, 2002.

[10] NIST/SEMATECH e-Handbook of Statistical Methods, http://www.itl.nist.gov/div898/handbook/, Sep 12, 2007

[11] Agrawal, R., Imielinski, T., and Swami, A. Mining association rules between sets of items in large databases. In Proceedings of ACM SIGMOD Conference on Management of Data, pp. 207-216, Washington, D.C., USA 1993.

[12] Agrawal, R., and Srikant, R. Fast Algorithms for Mining Association Rules. In Proceedings of the 20^{th} International Conference on Very Large Data Bases, pp. 487-499, Santiago de Chile, Chile, September 1994.

[13] Ordonez, C., Santana, C., and Braal, D. Discovering Interesting Association Rules in Medical Data. ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery, pp. 78-85, Dallas, Texas, USA, 2000.

[14] Ordonez, C., Ezquerra, N., and Santana, C. A. Constraining and summarizing association rules in medical data. Knowledge and Information Systems 9(3), pp 1-2, 2006.

[15] Agarwal, R. C., Aggarwal, C. C., and Prasad, V. V. A tree projection algorithm for generation of frequent item sets. Journal of Parallel Distributed Computing 61(3), pp. 350-371, March, 2001.

[16] Shen, B., Yao, M., Wu, Z., Zhang, Y., and Y, W. Ontology-based Association Rules Retrieval using Protege Tools. IEEE International Conference on Data Mining (ICDM) Workshops, pp. 765 - 769, Hong Kong, China, 2006.

[17] Zaki, M.J. Parallel and Distributed Association Mining: A Survey. IEEE Concurrency, special issue on Parallel Mechanisms for Data Mining, 7(4), pp. 14-25, 1999.

[18] Zaki, M. J. Scalable Algorithms for Association Mining. IEEE Transactions on Knowledge and Data Engineering 12(2), pp, 372-390, 2000.

[19] Shenoy, P., Haritsa, J. R., Sundarshan, S., Bhalotia, G., Bawa, M., and Shah, D. Turbo-charging vertical mining of large databases. SIGMOD Rec. 29(2), pp. 22-33, June, 2000.

[20] Orlando, S., Palmerini, P., Perego, R., and Silvestri, F. Adaptive and Resource-Aware Mining of Frequent Sets. In Proceedings of the 2002 IEEE international Conference on Data Mining, pp. 338-345, Washington, DC, USA, 2002.

[21] Park, J. S., Chen, M., and Yu, P. S. An effective hash-based algorithm for mining association rules. In Proceedings of the ACM SIGMOD international Conference on Management of Data, pp. 175-186, San Jose, California, USA, 1995.

[22] Brin, S., Motwani, R., Ullman, J. D., and Tsur, S. Dynamic itemset counting and implication rules formarket basket data.In Proceedings of the ACM SIGMOD international Conference on Management of Data, pp. 255-264, Tucson, Arizona, USA, 1997.

[23] Savasere, A., Omiecinski, E., and Navathe, S. B. An Efficient Algorithm for Mining Association Rules in Large Databases. In Proceedings of the 21th international Conference on Very Large Data Bases, pp. 432-444, Zurich, Switzerland, 1995.

[24] J. Han, J. Pei, Y. Yin and R. Mao. Mining Frequent Patterns without Candidate Generation: A Frequent-Pattern Tree Approach. Data Mining and Knowledge Discovery, 8(1), pp. 53-87, 2004.

[25] Pei, J., Han, J., Lu, H., Nishio, S., Tang, S., and Yang, D. H-Mine: Hyper-Structure Mining of Frequent Patterns in Large Databases. In Proceedings of IEEE international Conference on Data Mining, pp 441-448, Washington, DC, USA, 2001.

[26] Gopalan, R. P. and Sucahyo, Y. G. TreeITL-Mine: Mining Frequent Itemsets Using Pattern Growth, Tid Intersection, and Prefix Tree. In Proceedings of the 15th Australian Joint Conference on Artificial intelligence: Advances in Artificial intelligence, pp. 535-546, 2002.

[27] Sucahyo, Y. G. and Gopalan, R. P. CT-ITL: efficient frequent item set mining using a compressed prefix tree with pattern growth. In Proceedings of the 14th Australasian Database Conference, pp. 95-104, Australia, 2003.

[28] Grahne, G., and Zhu, J. Efficiently Using the Prefix-trees in Mining Frequent Itemsets. FIM'03, Workshop on Frequent Itemset Mining Implementations. Melbourne, Florida, USA, 2003.

[29] EL-Hajj, M., and Zaiane, O.R. Non Recursive Generation of Frequent K-itemset from Frequent Pattern Tree Representation. FIMI'03, Workshop on Frequent Itemset Mining Implementations. Melbourne, Florida, USA, November 2003.

[30] Frequent Itemset Mining Implementaions, FIMI04, http://fimi.cs.helsinki.fi/fimi04/. December 27, 2007.

[31] http://www.adrem.ua.ac.be/ goethals/software. May, 2006.

[32] Agrawal, R. and Srikant, R. Mining Sequential Patterns. In Proceeding of the 11th International Conference Data Engineering (ICDE 95), pp. 3-14, Taipei, 1995.

[33] Pei, J., Han, J., Mortazavi-Asl, M., Wang, J., Pinto, H., Chen, Q., Dayal, U., and Hsu, M. Mining Sequential Patterns by Pattern-Growth: The PrefixSpan Approach. In proceeding of IEEE Transactions on Knowledge and Data Engineering. 16(11), pp. 1424-1440, 2004.

[34] Kosala, R. and Blockeel, H. Web mining research: a survey. SIGKDD Explor. Newsl. 2(1), pp. 1-15, 2000.

[35] Wang, J., Zaki, M.J., Toivonen, H., Shasha, D.(Eds.). Data Mining in Bioinformatics. Springer London Ltd, 2004.

[36] Hu, Y. and Panda, B. A data mining approach for database intrusion detection. In Proceedings of the 2004 ACM Symposium on Applied Computing, pp. 711-716, Nicosia, Cyprus, 2004.

[37] Mannila, H., Toivonen, H., and Verkamo, A.I. Discovering frequent episodes in sequences. In Proceedings of the First International Conference on Knowledge Discovery and Data Mining [KDD 95], pp. 210-215, Montreal, Canada, August 1995.

[38] Boukerche, A., Silva, F. H., Araujo, R. B., and Pazzi, R. W. A low latency and energy aware event ordering algorithm for wireless actor and sensor networks. In Proceedings of the 8th ACM international Symposium on Modeling, Analysis and Simulation of Wireless and Mobile Systems, pp. 111 - 117, Montral, Canada, 2005.

[39] Römer, K. Temporal Message Ordering in Wireless Sensor Networks. IFIP Mediterranean Workshop on Ad-Hoc Networks. pp. 131-142, Mahdia, Tunisia, 2003.

[40] Tan, P-N. Knowledge Discovery from Sensor Data. Sensors 23(3), pp. 14-19, 2006.

[41] Babcock, B., Babu, S., Datar, M., Motwani, R., and Widom, J. Models and issues in data stream systems. In Proceedings of the Twenty-First ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems. pp. 1-16, Madison, Wisconsin, 2002.

[42] Golab, L. and Özsu, M. T. Issues in data stream management. SIGMOD Rec. 32(2), pp 5-14. June, 2003.

[43] Radivojac, P., Korad, U., Sivalingam, K. M., and Obradovic, Z. Learning from class-imbalanced data in wireless sensor networks. IEEE Semiannual Vehicular Technology Conference, Vol. 5, pp. 3030-3034. Orlando, Florida, USA, 2003.

[44] Japkowicz, N. The Class Imbalance Problem: Significance and Strategies. In Proceedings of the 2000 International Conference on Artificial Intelligence (IC-AI'2000), pp. 111-117. 2000.

[45] McConnell, S., and Skillicorn D. A Distributed Approach for Prediction in Sensor Networks. In Proceedings of 1st International Workshop on Data Mining in Sensor Networks, pp. 28-37, 2005.

[46] Duarte, M.F, and Hu, Y.H. Vehicle Classification in Distributed Sensor Networks. Journal of Parallel and Distributed Computing, 64(7), pp. 826-838, 2004.

[47] Gu, L., Jia, D., Vicaire, P., Yan, T., Luo, L., Tirumala, A., Cao, Q., He, T., Stankovic, J. A., Abdelzaher, T., and Krogh, B. H. Lightweight detection and classification for wireless sensor networks in realistic environments. In Proceedings of the 3rd international Conference on Embedded Networked Sensor Systems, pp. 205- 217, San Diego, California, USA, November, 2005.

[48] Wang, X., Bi, D., Ding, L., and Wang, S. Agent Collaborative Target Localization and Classification in Wireless Sensor Networks. Sensors 7(8), pp. 1343-1358, 2007.

[49] Krishnamachari, B. and Iyengar, S. Distributed Bayesian Algorithms for Fault-Tolerant Event Region Detection in Wireless Sensor Networks. IEEE Transaction on Computers 53(3), pp. 241-250, March, 2004.

[50] Lazaridis, I., and Mehrotra, S. Capturing Sensor-Generated Time Series with Quality Quarantee. In Proceedings of the 19th International Conference on Data Engineering (ICDE'03). Bangalore, India, 2003.

[51] Goel, S., and Imielinski, T. Prediction-based Monitoring in Sensor Networks: Taking Lessons from MPEG. ACM Computer Communication Review, 31(5), pp. 82-98, October, 2001.

[52] Kalman, R. E. A New Approach to Linear Filtering and Prediction Problems. Transaction of the ASME-Journal of Basic Engineering, 82(series D), pp. 35-45, 1996.

[53] Jain, A., Chang, E. Y., and Wang, Y. Adaptive stream resource management using Kalman Filters. In Proceedings of the 2004 ACM SIGMOD international Conference on Management of Data, pp. 11-22, Paris, France, 2004.

[54] Olston, C., Jiang, J., and Widom, J. Adaptive filters for continuous queries over distributed data streams. In Proceedings of the 2003 ACM SIGMOD international Conference on Management of Data, pp. 563-574, San Diego, California, USA, 2003.

[55] Le Borgne, Y., Santini, S., and Bontempi, G. Adaptive model selection for time series prediction in wireless sensor networks. Signal Process. 87(12), pp 3010-3020, December, 2007.

[56] Chu, D., Deshpande, A., Hellerstein, J. M., and Hong, W. Approximate Data Collection in Sensor Networks using Probabilistic Models. In Proceedings of the 22nd international Conference on Data Engineering, pp 48-60, April, 2006.

[57] Tulone, D., and Madden, S. PAQ: time series forecasting for approximate query answering in sensor networks. In the 3rd European Workshop on Wireless Sensor Networks, pp. 21-37, Zurich, Switzerland, 2006.

[58] Deshpande, A., Guestrin, C., Madden, S. R., Hellerstein, J. M., and Hong, W. Model-driven data acquisition in sensor networks. In Proceedings of the 30th Very Large Data Bases conference, pp. 588-599. Toronto, Canada, 2004.

[59] Santini, S., and Römer, K. An adaptive strategy for quality-based data reduction in wireless sensor networks. In proceedings of the 3rd International Conference on Networked Sensing Systems (INSS06), pp. 29-36, Chicago, IL, USA, 2006.

[60] Gedik, B., Liu, L., and Yu, P. S. 2007. ASAP: An Adaptive Sampling Approach to Data Collection in Sensor Networks. IEEE Trans. Parallel Distrib. Syst. 18(12), pp. 1766-1783, December, 2007.

[61] Haykin, S., and Widrow, B. Least-Mean-Square Adaptive Filters. John Wiley Sons, 2003.

[62] Yann-Ael, L.B., and Gainluca, B. Round robin cycle for predictions in wireless sensor networks. In 2nd International Conference on Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP'05), pp. 253-258, Melbourne, Australia, 2005.

[63] Mini, R. A., Machado, M. D. V., Loureiro, A. A., and Nath, B. Prediction-based energy map for wireless sensor networks. Ad Hoc Networks 3(2005), pp. 235253, 2005.

[64] Mini, R. A., Loureiro, A. A., and Nath, B. Energy map construction for wireless sensor network under a finite energy budget. In Proceedings of the 7th ACM international Symposium on Modeling, Analysis and Simulation of Wireless and Mobile Systems, pp. 165-169, Venice, Italy, October, 2004.

[65] Bontempi, G., and Borgne, Y.-A. An Adaptive Modular Approach to the Mining of Sensor Network Data. In Proceedings of 1st International Workshop on Data Mining in Sensor Networks as part of the SIAM International Conference on Data Mining, pp. 3-9, 2005.

[66] Smith, L.I. A tutorial on Principal Components Analysis. 2006. Available at *http : //csnet.otago.ac.nz/cosc*453/*student_tutorials/principal_components.pdf*, (Oct 31, 2007).

[67] Birattari, M., Bontempi, G., and Bersini, H. Lazy learning meets the recursive least squares algorithm. In Proceedings of the 1998 Conference on Advances in Neural information Processing Systems II D. A. Cohn, Ed. MIT Press, Cambridge, MA, pp. 375-381, 1989.

[68] Hwang, S., Su, Y., Lin, Y., and Dow, C. A Cluster-Based Coverage-Preserved Node Scheduling Scheme in Wireless Sensor Networks. In the 3rd Annual International Conference on Mobile and Ubiquitous Systems - Workshops, pp. 1-7, 2006.

[69] Tian, D. and Georganas, N. D. A coverage-preserving node scheduling scheme for large wireless sensor networks. In Proceedings of the 1st ACM international Workshop on Wireless Sensor Networks and Applications, pp. 32-41. Atlanta, Georgia, USA, 2002.

[70] Huang, C. and Tseng, Y. The coverage problem in a wireless sensor network. In Proceedings of the 2nd ACM international Conference on Wireless Sensor Networks and Applications, pp. 115-121, San Diego, CA, USA, September, 2003.

[71] Chen, J. and Koutsoukos, X. Survey on Coverage Problems in Wireless Ad Hoc Sensor Networks. IEEE SouthEastCon 2007. Richmond, VA, March, 2007.

[72] Loo, K. K., Tong, I., Kao, B., and Chenung, D. Online Algorithms forMining Inter-Stream Associations From Large Sensor Networks. In Proceedings of the Ninth Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD05), pp. 143-149. Hanoi, Vietnam, May 2005.

[73] Manku, G. S., and Motwani, R. Approximate Frequency Counts over Streaming Data. In Proceedings of the 28th International Conference on Very Large Data Bases, pp. 346-357, Hong Kong, August, 2002.

[74] Romer, K. Distributed Mining of Spatio-Temporal Event Patterns in Sensor Networks. In Proceeding of Euro-American Workshop on Middleware for Sensor Networks /DCOSS, pp. 103-116, San Francisco, USA, June 2006.

[75] Cantoni, V., Lombardi, L., and Lombardi, P. Challenges for Data Mining in Distributed Sensor Networks. In Proceedings of the 18th international Conference on Pattern Recognition (Icpr'06), pp. 1000-1007, August, 2006.

[76] Bodon, F., Schmidt-Thieme L. The Relation of Closed Itemset Mining, Complete Pruning Strategies and Item Ordering in APRIORI-based FIM algorithms. The 9th European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD), pp. 437-444, Porto, Portugal, 2005.

[77] Gouda, K., and Zaki, M. J. Efficiently Mining Maximal Frequent Itemsets. In Proceedings of the IEEE international Conference on Data Mining, pp. 163-170, November, 2001.

[78] Halatchev, M., and Gruenwald, L. Estimating Missing Values in Related Sensor Data Streams. In Proceedings of the 11th International Conference on Management of Data, pp. 83-94, India, January. 2005.

[79] Davidson, I., and Ravi, A. Distributed Pre-Processing of Data on Networks of Berkeley Motes Using Non-Parametric EM. In Proceedings of 1st International Workshop on Data Mining in Sensor Networks, pp. 17-27, 2005.

[80] Leung, C.K., Khan, Q.I., Hao, B. Distributed Mining of Constrained Patterns from Wireless Sensor Data. IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology (WI-IATW'06), pp, 248-251, 2006.

[81] Akyildiz, I.F., Su, W., Sankarasubramaniam, Y., and Cayirci, E. Wireless sensor networks: a survey. Computer Networks, vol 38(4), pp 393-422, March, 2002.

[82] Akkaya, K.,Younis, M. A Survey of Routing Protocols in Wireless Sensor Networks. Elsevier Ad Hoc Network Journal, 3(3), pp. 325-349, 2005.

[83] Akyildiz, I., and Kasimoglu, I. Wireless sensor and actor networks: Research challenges. Ad Hoc Networks Journal (Elsevier), 2(4), pp 351-367, October, 2004.

[84] Krishnamachari, B., Estrin, D., and Wicker, S. Modeling Data Centric Routing in Wireless Sensor Networks. In the Proceedings of IEEE INFOCOM, New York, USA, June 2002.

[85] Chong, C.Y., and Pumar, S. Sensor networks: Evolution, opportunities, and challenges. In Proceedings of the IEEE vol 91(8), pp. 12471256, Aug. 2003.

[86] Subramanian, L. and Katz, R. H. An architecture for building self-configurable systemsIn Proceedings of the 1st ACM international Symposium on Mobile Ad Hoc Networking and Computing, pp. 63-73. NJ, USA, 2000.

[87] N. Noury, T. Herve, V. Rialle, G. Virone, E. Mercier, G. Morey, A. Moro, T. Porcheron. Monitoring behavior in home using a smart fall sensor, IEEE-EMBS Special Topic Conference on Microtechnologies in Medicine and Biology, pp. 607-610, October, 2000.

[88] Milenkovic, A., Otto, C., and Jovanov, E. Wireless Sensor Networks for Personal Health Monitoring: Issues and an Implementation. Computer Communications. Vol 29, No. 13-14, pp. 2521-2533, 2006.

[89] Aziz, O., Lo, B., King, R., Yang, G., and Darzi, A. Pervasive Body Sensor Network: An Approach to Monitoring the Post-operative Surgical Patient. International Workshop on Wearable and Implantable Body Sensor Networks, pp.1318, 2006.

[90] Murthy, C.S.R, Manoj, B.S. Ad Hoc Wireless Networks Architecture and Protocols. Prentic Hall PTR, New Jersey, 2004.

[91] Mainwaring, A., Culler, D., Polastre, J., Szewczyk, R., and Anderson, J. Wireless sensor networks for habitat monitoring. In Proceedings of the 1st ACM international Workshop on Wireless Sensor Networks and Applications, pp 88-97, Atlanta, USA, 2002.

[92] Tilak, S., Abu-Ghazaleh, N. B., and Heinzelman, W. A taxonomy of wireless micro-sensor network models. Mobile Computing and Communications Review(9), pp. 28-36, 2002.

[93] Römer, K., Mattern, F., and Zurich, E. The Design Space of Wireless Sensor Networks. IEEE Wireless Communications vol. 11(6), pp. 54-61, 2004.

[94] Heinzelman, W. R., Chandrakasan, A., and Balakrishnan, H. Energy-Efficient Communication Protocol for Wireless Microsensor Networks. In Proceedings of the 33rd Hawaii international Conference on System Sciences, pp. 8020-8030. January, 2000.

[95] Heinzelman, W. R., Kulik, J., and Balakrishnan, H. Adaptive protocols for information dissemination in wireless sensor networks. In Proceedings of the 5th Annual ACM/IEEE international Conferenceon Mobile Computing and Networking, pp. 174-185, Washington, USA, August 1999.

[96] He, T., Stankovic, J. A., Lu, C., and Abdelzaher, T. SPEED: A Stateless Protocol for Real-Time Communication in Sensor Networks. In Proceedings of the 23rd international Conference on Distributed Computing Systems, pp. 46-55, 2003.

[97] Mathur, G., Desnoyers, P., Ganesan, D., and Shenoy, P. Ultra-Low Power Data Storage for Sensor Networks. In Proceedinig of the Fifth IEEE/ACM Conference on Information Processing in Sensor Networks, pp. 374- 381. Nashville TN, April , 2006.

[98] Toshiba 128-MBIT ($16M8BITS/8Mx16BITS$) CMOS NAND E2PROM. $http : //www.tranzistoare.ro/datasheets2/37/378494_1.pdf$ Datasheet: TC58DVM72A1FT00, Feb. 2008.

[99] Desnoyers, P., Ganesan, D., Li, H., and Shenoy, P. PRESTO: A predictive storage architecture for sensor networks. In the Tenth Workshop on Hot Topics in Operating Systems (HotOS X), pp. 23-28, June 2005.

[100] Madden, S., Franklin, M. J., Hellerstein, J. M., and Hong, W. TAG: a Tiny AGgregation service for ad-hoc sensor networks. ACM SIGOPS Operating Systems Review, Volume 36, Special Issue, pp 131-146, 2002.

[101] http://www.mathworks.com/products/matlab/, Feb. 2008.

[102] Intel Lab Data, http://berkeley.intel-research.net/labdata/, June. 2007.

[103] Eugster, P. T., Felber, P. A., Guerraoui, R., and Kermarrec, A. The many faces of publish/subscribe. ACM Comput. Surv. 35(2), pp 114-131, June, 2003.

[104] Intanagonwiwat, C., Govindan, R., Estrin, D., Heidemann, J., and Silva, F. Directed diffusion for wireless sensor networking. IEEE/ACM Transactions on Networking (TON), 11(1), pp. 2-16, Feb, 2003.

[105] Braginsky, D., Estrin, D. Rumor routing algorithm for sensor networks. In Proceedings of the First ACM International Workshop on Wireless Sensor Networks and Applications, pp. 22-31, Atlanta, GA, September 2002.

[106] Shah, R. C., and Rabaey, J. M. Energy aware routing for low energy ad-hoc sensor networks. In Proceedings of the 3rd IEEE Wireless Communications and Networking Conference, pp. 635-644, Orlando, USA, 2001.

[107] Schurgers, C., and Srivastava, M.B. Energy efficient routing in wireless sensor networks. In the Proceedings of IEEE MILCOM on Communications for Network-Centric Operations: Creating the Information Force, pp. 357-361, McLean, VA, 2001.

[108] Chu, M., Haussecker, H., Zhao, F. Scalable Information-Driven Sensor Querying and Routing for ad hoc Heterogeneous Sensor Networks. The International Journal of High Performance Computing Applications, 16(3), August 2002.

[109] Lindsey, S., and Raghavendra, C.S. PEGASIS: Power Efficient GAthering in Sensor Information Systems. In the proceedings of the IEEE Aerospace Conference, pp 1125-1130, March 2002.

[110] Lindsey, S., Raghavendra, C., and Sivalingam, K. M. Data Gathering Algorithms in Sensor Networks Using Energy Metrics. IEEE Transactions on Parallel and Distributed Systems vol. (13)9, pp. 924- 935, 2002.

[111] Xu, Y., Heidemann, J., and Estrin, D. Geography-informed energy conservation for Ad Hoc routing. In Proceedings of the 7th Annual international Conference on Mobile Computing and Networking Rome, pp. 70-84, Italy, 2001.

[112] Rodoplu, V., and Ming, T.H. Minimum energy mobile wireless networks. IEEE Journal of Selected Areas in Communications. 17(8), pp. 1333-1344, 1999.

[113] Sohrabi, K., Gao, j., Ailawadhi, V., and Pottie, G.J. Protocols for Self-Organization of a Wireless Sensor Network. IEEE personal Communcation 7(5), pp. 16-27, 2000.

List of Publications

- Azzedine Boukerche and Samer Samarah: A Novel Algorithm for Mining Association Rules in Wireless Ad-hoc Sensor Networks. *IEEE Transactions on Parallel and Distributed Systems*, pp. 865-877, 19(7), 2008.

- Samer Samarah and Azzedine Boukerche. Chronological Tree-A Compressed Structure for Mining Behavioral Patterns in Wireless Sensor Networks. *Journal of Interconnected Networks.* To appear in Fall 2008.

- Azzedine Boukerche and Samer Samarah. Coverage-based Sensor Association Rules. *IEEE Transactions on Parallel and Distributed Systems.* (Submitted)

- Azzedine Boukerche, Samer Samarah. Knowledge Discovery in Wireless Sensor Networks for Chronological Patterns. *IEEE Conference on Local Computer Networks*, 2008.

- Samer Samarah and Azzedine Boukerche. Coverage-based Sensor Association Rules for Wireless Vehicular Ad hoc and Sensor Networks. *IEEE Globecom*, 2008.

- Azzedine Boukerche and Samer Samarah. A Performance Evaluation of Distributed Framework for Mining Wireless Sensor Networks. In Proceeding of the 40^{th} *Annual Simulation Symposium*, pp. 239-246, 2007.

- Azzedine Boukerche and Samer Samarah. A New Representation Structure for Mining Association Rules from Wireless Sensor Networks. In Proceeding of IEEE *Wireless Communications and Networking (WCNC)*, pp. 2855-2860, 2007.

- Azzedine Boukerche and Samer Samarah. An Efficient Distributed Data Extraction Algorithm for Mining Association Rules from Wireless Sensor Networks. In Proceeding of IEEE International Conference on Communications (ICC), pp. 3936-3941, 2007

- Azzedine Boukerche and Samer Samarah. A New Mechanism for In-network Data Reduction to Extract Data for Mining Wireless Sensor Networks. In Proceedings of the 10^{th} *ACM Symposium on Modeling, Analysis, and Simulation of Wireless and Mobile Systems*, pp. 70-77. 2007.

- Azzedine Boukerche and Samer Samarah. A Novel Data Mining Technique for Extracting Events and Inter Knowledge based Information from Wireless Sensor Networks. In Proceeding of 31st IEEE *Conference on Local Computer Networks (LCN)*, pp. 769-775. 2006.

- Azzedine Boukerche and Samer Samarah. PLT- Positional Lexicographic Tree: A New Structure for Mining Frequent Itemsets. *ICPP* Workshops, pp. 135-141. 2006.