# DATA WAREHOUSING FOR SOCIAL NETWORKING AND HUMAN MOBILITY RESEARCH AND DETECTING REAL-WORLD EVENTS IN SPACE AND TIME USING FEATURE CLUSTERING

A Dissertation

Submitted to the Graduate School

of the University of Notre Dame

in Partial Fulfillment of the Requirements

for the Degree of

Doctor of Philosophy

by

Alec Pawling

---

Greg Madey, Director

Graduate Program in Computer Science and Engineering

Notre Dame, Indiana

December 2010

# DATA WAREHOUSING FOR SOCIAL NETWORKING AND HUMAN MOBILITY RESEARCH AND DETECTING REAL-WORLD EVENTS IN SPACE AND TIME USING FEATURE CLUSTERING

Abstract

by

Alec Pawling

In this dissertation, we address two problems associated with the development of an emergency response system that utilizes a cell phone network as a sensor network to facilitate the initiation and online validation of predictive simulations. Schoenharl [77] describes the simulation component of the emergency response system; this dissertation addresses two other significant components of the system: the historical data source and the detection and alert system.

The historical data source is a data warehouse designed to facilitate the development of additional simulation models for the predictive component of the system and has wider applications for scientific research on social networks and human mobility.

The detection and alert system provides an automatic mechanism for initiating the simulation system without intervention by emergency response managers. This system identifies potential emergency events in both time and space, allowing the simulations to begin shortly after an event as well as focus on the area affected by the event, reducing the computational costs significantly.

CONTENTS

FIGURES

TABLES

## ACKNOWLEDGMENTS

This material is based upon work supported by the National Science Foundation under Grant Nos. CNS-0540348 and BCS-0826958, Northeastern University, and the Defense Threat Reduction Agency under Grant BRBAA07-J-2-0035.

# CHAPTER 1

# INTRODUCTION

In this dissertation, we address two problems associated with the development of an emergency response system that utilizes a cell phone network as a sensor network to facilitate the initiation and online validation of predictive simulations. Schoenharl [77] describes the simulation component of the emergency response system; this dissertation addresses two other significant components of the system: the historical data source and the detection and alert system.

The historical data source is a data warehouse designed to facilitate the development of additional simulation models for the predictive component of the system and has wider applications for scientific research on social networks and human mobility. In this dissertation, we consider two aspects of the data warehouse in detail: 1) the design, which, given the way in which the data is provided, is treated as a data reduction and partitioning problem to facilitate efficient access, and 2) a data cleaning problem which addresses the necessary issues to permit us to assess the meaningfulness of merging multiple, noisy data sets that, in theory, describe the same set of interactions. In practice, we discover that noise in the data introduces redundant records when merged.

The detection and alert system provides an automatic mechanism for initiating the simulation system without intervention by emergency response managers. This system identifies potential emergency events in both time and space, allowing the simulations to begin shortly after an event and focus on the area affected by the event, significantly reducing the computational costs. We present two clustering algorithms implemented for the detection and alert system. The first algorithm, a one pass, incremental algorithm, is not able to detect events in the data; however, given the correct parameterization, it produces a model of a data set of comparable quality to an offline algorithm. The second algorithm, a feature clustering algorithm over a sliding window, allows us to detect events that results in a spike in call activity, and, when linked with the information used to generate the data set, gives a spatial location of the event.

## 1.1 Organization

The remainder of this chapter provides the context for the research presented in this document. We discuss the motivating application, the Wireless Phone Based Emergency Response System, and it's underlying concept, Dynamic Data-Driven Application Systems.

The remainder of the document is split into two parts: the first part describes the historical data source. Chapter 2 provides a background in database management systems and data warehousing relevant to the development of the historical data source. Chapter 3 discusses the significant aspects of the design of the historical data source, often motivated by the way the data has been used in research in the past. Chapter 4 describes an approach for matching redundant records introduced due to noise when merging the various aspects of the cell phone usage data.

The second part describes the detection and alert system. Chapter 5 gives a background in data clustering relevant to the development of the detection and alert system. Chapter 6 describes a novel one-pass incremental clustering algorithm. Chapter 7 describes an approach for event detection using feature clustering over a sliding window of time series, each associated with a spatial location. Using this approach, we identify anomalies in a subset of the time series, which indicate the location of interest.

## 1.2 Background

### 1.2.1 The Wireless Phone Based Emergency Response System

The Wireless Phone-Based Emergency Response (WIPER) system is a proof-of-concept prototype designed to utilize a cell-phone network as a set of sensors for gathering and presenting data to emergency response managers. The system would monitor the network data in real time for anomalous activity, run simulations to predict population movement during a crisis, and provide emergency response managers with a current view of the affected area using GIS tools [56, 57, 78, 79] [1].

Existing software tools, such as EVResponse [88] and COMBINED [86] provide mechanisms for manually gathering information relating to the current status of a crisis situation. There is a high cost associated with such systems in terms of time and money. Wireless devices and network infrastructure must be purchased to facilitate data collection, personnel must be trained to use the technology, and personnel must be deployed to the affected area to collect the data. In contrast, WIPER provides information about the situation through a pre-existing network, requiring no investment in infrastructure or deployment; however, we gain these advantages at the cost of data flexibility.

To counteract the limitations of the data, the WIPER system is designed to use machine learning methods to generate hypotheses about the causes of anomalies detected in the data. These hypotheses are tested using a dynamic data driven application system of simulations. Dynamic data driven application systems (DDDAS) are characterized by an ability to incorporate new data into running simulations. Research in DDDAS is

[1] Portions of this section were published in the Proceedings of the 5th International ISCRAM Conference [67]

motivated by a need for greater accuracy in complex simulations, e.g. simulations for predicting weather or wildfire propagation [22]. Once the WIPER system detects an anomaly, it will start a suite of simulations based on the hypotheses generated. These simulations will be validated using new data as it becomes available, allowing simulations that do not reflect the real world situation to be discarded.

In addition to the prediction capabilities, WIPER will provide the ability to view the development of a crisis in real-time, the ability to propose and evaluate responses in near real-time, and the ability to collect and analyze streaming information from a cellular communication network. The WIPER system will analyze dynamic data from the cell phone network in real-time, providing the functionality to detect crises as they emerge. Responding to events from the anomaly detection system, GIS-based simulations of the region will be launched and results collated and presented to planners. Finally, the web-based console will allow emergency response managers to quickly examine the current state of the environment, see predicted outcomes from the simulations, and evaluate possible courses of action.

The WIPER system consists of five components, each of which is described briefly below.

- The Decision Support System (DSS) is a web-based front end through which emergency response managers interact with the WIPER system.

- The Detection and Alert System (DAS) monitors streaming network data for anomalous activity. There are various aspects of the cell-phone network data that may be of interest, including overall usage levels, spatial distribution of the call activity, and the underlying social network.

- The Simulation and Prediction System (SPS) receives anomaly alerts from the DAS, produces hypotheses that describe the anomaly, and uses simulations in conjunction with streaming activity data to validate or reject hypotheses.

- The Historical Data Source (HIS) is a repository of cell phone network data that resides in secondary storage. This data is used to determine the base-line behavior of the network against which anomalies are detected and to periodically calibrate and update the DAS.

- The Real-Time Data Source (RTDS) is designed to receive transaction data directly from a cellular service provider. The RTDS is responsible for handling requests for streaming data from the DAS, SPS, and DDS and streaming incoming data to these components in real-time.

Figure 1 shows an architectural overview of the WIPER system. The RTDS and HIS will provide the bridge from the service provider and the WIPER system. The figure shows the flow of streaming data from the service provider through the RTDS, possibly by way of the HIS for development and training, and to the remaining components. Requests for streaming data from the RTDS occur via SOAP messages. SOAP messages are also used by the DAS to inform the SPS of potential anomalies in the streaming data.

In this dissertation, we develop two components of the WIPER system: the historical data source, in the form of a data warehouse, and the detection and alert system.

### 1.2.2 Dynamic Data Driven Application Systems

Dynamic data driven application systems (DDDAS) simulations and models make use of new data (typically provided by a sensor network) as it becomes available. This data validates and steers running simulations in order to improve their quality. Dynamic data driven application systems typically consist of three components: a simulation

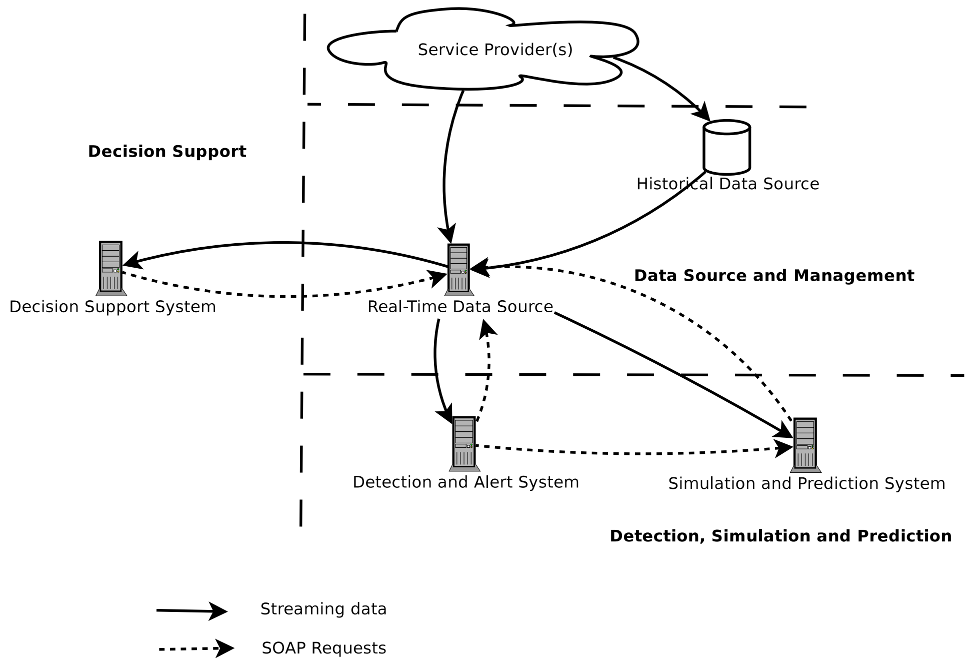


Figure 1.1. WIPER system architecture.

system, a sensor network, and a software system that provides data from the sensor network to the simulation system as requested [20].

Darema [20] identifies four areas of research that are necessary for the development of DDDAS applications:

- Application simulations must be able to utilize new data dynamically at runtime.
- Mathematical algorithms that ensure stability and robustness when subjected to dynamic inputs.
- Systems software provide fault tolerance and a certain level of quality of service to ensure that the applications are making sufficient progress (several DDDAS applications must run faster than real time), and that the applications can receive the streaming data required to improve accuracy.

- Interfaces to measurement infrastructure, such as wireless sensor networks, for management of data collection.

There are several DDDAS applications related to the problem of emergency response under development. We briefly describe two of these applications below.

The Vehicle-Infrastructure Integration initiative uses a set of roadside and in-vehicle sensors to monitor traffic flow and evaluate the highway system. The system also provides emergency detection and response support by detecting changes in flow of traffic, even in cases where part of the sensor network is lost, and running distributed simulations in support of a dynamic emergency response plan [30].

The iRevive system is intended to help emergency responders in the triage phase of mass casualty emergencies. The system uses data received through wireless hand-held computers carried by emergency responders and sensor that monitor patient vital signs to help the responders determine the order in which patients are treated [31].

Additionally, DDDAS application systems have been applied to problems such as agent based simulation [80], wildfire forecasting [58], simulation of fire spreading in a building [14], hurricane forecasting [4], and weather forecasting [71].

# CHAPTER 2

# BACKGROUND ON DATABASE MANAGEMENT SYSTEMS AND DATA WAREHOUSING

In this chapter, we provide background information on database management systems and data warehousing. Database management systems traditionally provide transaction processing in a manner that protects data consistency for a particular operation, a business for instance. In the first section of this chapter, we discuss the elements of database management systems that are relevant to the implementation of the historical data source. More recently, database management systems have provided a foundation for data analysis associated with decision support in businesses. The long term data required for this type of analysis is stored in a data warehouse, which is discussed in the second part of this chapter.

## 2.1 Database Management Systems

Database management systems provide a number of advantages over using traditional file storage, including concurrency control, data protection via transaction logging, and efficient data extraction and manipulation in a manner that is transparent to the user or application developer.

### 2.1.1 Introduction to the Relational Model

The relational data models used in database today is first described by Codd in a seminal paper [18]. The model was introduced as a foundation for addressing a number of existing problems in information systems at the time. Several of these problems related to data dependencies that have the potential to create problems when the underlying data storage format changes. These dependencies include those that arise when information systems rely on a particular data ordering, the presence of indexes on particular elements of the data, and the underlying organization of the files storing the data.

The model describes data in terms of relations. Relation $\mathbf{R}$ is a set of $n$-tuples where each element of the tuples are drawn from a set of sets $S_1, S_2, \ldots, S_n$ which define the domains of each column, or field. Let $r \in R$ be a row of the relation. A row can be expressed as $r = (s_1, s_2, \ldots, s_n)$ where $s_1 \in S_1, s_2 \in S_2, \ldots s_n \in S_n$, or, equivalently, $r \in S_1 \times S_2 \times \cdots \times S_n$. Along with this definition of a relation, certain properties are specified: each row in the relation must be distinct and no assumptions can be made about the order in which the rows are stored [18].

With the requirement that each row is distinct, a mechanism for identifying each row is specified. A *primary key*, defined as a set of fields serves as a unique identifier for the records in a relation, serves this purpose. The primary key can be referenced by other relations to link the two relations. Suppose $s_i$ is a primary key in relation $\mathbf{R}_1$, and $\mathbf{R}_2$ is a relation that contains some field $s_j$ that has the same domain as $s_i$ and is used to refer to records in $\mathbf{R}_1$. In this case we call $s_j$ a *foreign key* [18]. The definition of primary and foreign keys provide a mechanism for the decomposition of tables with compound attributes. The compound attributes can be moved into new relations after adding a foreign key that maps the new to the corresponding record in the original table

[18].

### 2.1.2 Key Constraints

The uniqueness of records, as described by Codd [18], is enforced by a *key constraint*, which is a minimal subset of fields that uniquely identifies a tuple in a relation. More generally, the set of *candidate keys* consists of all key constraints of a relation, and a *superkey* is a set of fields that is a super set of a candidate key [72].

More formally, a keys can be defined as a functional dependency. Suppose we have a relation **R** and that $\mathcal{X}$ and $\mathcal{Y}$ are two disjoint subsets of the fields in **R**. $\mathcal{Y}$ is *functionally dependent* on $\mathcal{X}$ if for two instances of $\mathcal{X}$, $\mathcal{X}_1$ and $\mathcal{X}_2$ and two instances of $\mathcal{Y}$, $\mathcal{Y}_1$ and $\mathcal{Y}_2$, $\mathcal{X}_1 = \mathcal{X}_2$ implies that $\mathcal{Y}_1 = \mathcal{Y}_2$. If $\mathcal{X}$ and $\mathcal{Y}$ are a partition of the fields in **R**, then $\mathcal{X}$ is a superkey. Functional dependencies are *trivial* in the case where $\mathcal{X}$ is a superset of $\mathcal{Y}$. Suppose that for a functional relation $\mathcal{X} \rightarrow \mathcal{Y}$ we remove one attribute in $\mathcal{X}$. If the functional dependency no longer holds, $\mathcal{Y}$ is said to be *fully functionally dependent* on $\mathcal{X}$; otherwise, it is a *partial functional dependency*. Furthermore, functional dependencies have the property of transitivity: let $\mathcal{X} \rightarrow \mathcal{Z}$ and $\mathcal{Z} \rightarrow \mathcal{Y}$ be functional dependencies, then $\mathcal{X} \rightarrow \mathcal{Y}$ is a *transitive functional dependency* [25].

### 2.1.3 Normalization

Database normalization provides a mechanism for eliminating data redundancy, which, if not eliminated, can lead to several problems:

- Update anomalies: This type of anomaly occurs when an update to a data item stored in multiple locations is not applied to all instances of that data item in the database. In this case, the database becomes inconsistent.

- Insert anomalies: This type of anomaly occurs when an instance of redundant data containing an error is added to the database.
- Delete anomalies: This type of anomaly occurs when the last instance of redundant data is removed from the database because the instance of the entity it is associated with is removed, even though it is beneficial to retain this data [25].

Normal forms enforce particular relationships between the primary key of a relation and the remaining attributes. First through third normal forms provide a progression of increasingly strict constraints on the nature of functional dependencies present in the relation: each form must conform to the requirements of all previous normal forms as well as its own particular constraint. First normal form only requires that each field of a relation consist of a single item. Second normal form requires that fields that are not part of the primary key be fully functionally dependent on the primary key. Third normal form forbids any field not in the primary key from being transitively functionally dependent on the primary key. Boyce-Codd normal form forbids elements of the primary key from being functionally dependent on any other set of fields [25].

### 2.1.4 Relational Algebra

Relational algebra is a formal query language on the relational model. It consists of two types of operations: set-theoretic operations and relation-theoretic operations. The set-theoretic operations are used to combine relations using the standard set operations: union ($\cup$), difference ($-$), intersection ($\cap$), and Cartesian product ($\times$). The union, difference, and intersection operations require *union compatibility* of the two relations, *i.e.* the two relations have the same number of fields and the corresponding fields are of the same type. The relation-theoretic operations constrain the set and fields that are returned by the query. The select operator, $\sigma_i$, returns the set of $n$-tuples for which a condition $i$ holds, and the project, $\pi_J$, operator constrains the members of each tuple returned to the set $J \subseteq \{R_1, R_2, \ldots, R_n\}$ [72, 84].

### 2.1.4.1 Relational Join Operations

The join operations extend the set operations to pairs of relations that are not union compatible, assuming that there is some subset of the two relations that are union compatible. There are two types of join operations: inner joins, which are analogous to set intersection; and outer joins, which are analogous to set union. Each join has an associated boolean condition, $\theta$, that gives the selection criteria for inner joins [25, 72].

Let $\mathbf{R} = (R_1, R_2, \ldots, R_{n_R})$ and $\mathbf{S} = (S_1, S_2, \ldots, S_{n_S}$ be two relations. The inner join of $\mathbf{R}$ and $\mathbf{S}$ gives a relation $(R_1, R_2, \ldots, R_{n_R}, S_1, S_2, \ldots, S_{n_S})$ such that some boolean condition $\theta$ on a union compatible subset of attributes of $\mathbf{R}$ and $\mathbf{S}$ holds, *e.g.* let $\theta$ be $R_1 = S_1 \wedge R_2 = S_2$ where $(R_1, R_2)$ and $(S_1, S_2)$ are union compatible attributes. The inner join, $\bowtie_\theta$, is, therefore, defined as a selection on a Cartesian product:

$$\mathbf{R} \underset{R_1=S_1}{\bowtie} \mathbf{S} = \sigma_{R_1=S_1}(\mathbf{R} \times \mathbf{S}) \tag{2.1}$$

[25, 72].

The outer join, $\times_\theta$, is a super set of the inner join where all of the records in $\mathbf{R}$ and $\mathbf{S}$ that are not in the inner join are added to the resulting relation padded with null values as necessary. Here we derive the relational algebra expression for an outer join following the description from [25]:

- We already know how to express the inner join:

$$R \underset{\theta}{\bowtie} S \tag{2.2}$$

- We compute set difference to find the elements in $R$ and $S$ that are not in the inner join:

$$R - \pi_{R_1,R_2,\ldots R_{n_R}}\left(R \underset{\theta}{\bowtie} S\right) \tag{2.3}$$

$$S - \pi_{R_1,R_2,\ldots,S_{n_S}}\left(R \underset{\theta}{\bowtie} S\right) \tag{2.4}$$

- We need to take the union of the two expressions above; however, we must first make them union compatible with the inner join by padding the set differences with $n_S$-tuples and $n_R$-tuples populated with null values, represented by $\omega$:

$$\left(R - \pi_{R_1,R_2,\ldots,R_{n_R}}\left(R \underset{\theta}{\bowtie} S\right)\right) \times \left(\underbrace{\omega, \omega, \ldots, \omega}_{n_S}\right) \tag{2.5}$$

$$\left(\underbrace{\omega, \omega, \ldots, \omega}_{n_R}\right) \times \left(S - \pi_{S_1,S_2\ldots,S_{n_S}}\left(R \underset{\theta}{\bowtie} S\right)\right) \tag{2.6}$$

- The outer join is the union of equations 2.2, 2.5, and 2.6:

$$\begin{aligned}\mathbf{R} \underset{\theta}{⟗} \mathbf{S} = &\left(\mathbf{R} \underset{\theta}{\bowtie} \mathbf{S}\right)\\ &\cup \left(\left(\mathbf{R} - \pi_{R_1,R_2,\ldots,R_{n_R}}\left(\mathbf{R} \underset{\theta}{\bowtie} \mathbf{S}\right)\right) \times \underbrace{(\omega, \omega, \ldots, \omega)}_{n_S}\right)\\ &\cup \left(\underbrace{(\omega, \omega, \ldots, \omega)}_{n_R} \times \left(\mathbf{S} - \pi_{S_1,S_2,\ldots,S_{n_S}}\left(\mathbf{R} \underset{\theta}{\bowtie} \mathbf{S}\right)\right)\right)\end{aligned} \tag{2.7}$$

There are two commonly used subsets of the outer join: the left outer join and right outer join. The left outer join is the union of the inner join of the two tables and the null padded rows of the left table that are not in the inner join:

$$\mathbf{R} \underset{\theta}{\ltimes} \mathbf{S} = \left(\mathbf{R} \underset{\theta}{\bowtie} \mathbf{S}\right) \cup \left(\left(\mathbf{R} - \pi_{R_1,R_2,\ldots,R_{n_R}}\left(\mathbf{R} \underset{\theta}{\bowtie} \mathbf{S}\right)\right) \times \underbrace{(\omega, \omega, \ldots, \omega)}_{n_{n_S}}\right) \tag{2.8}$$

The right outer join is the union of the inner join and the null padded rows of the right table that are not in the inner join:

$$\mathbf{R} \underset{\theta}{\rtimes} \mathbf{S} = \left(\mathbf{R} \underset{\theta}{\bowtie} \mathbf{S}\right) \cup \left(\underbrace{(\omega, \omega, \ldots, \omega)}_{n_{n_R}} \times \left(\mathbf{S} - \pi_{S_1,S_2,\ldots,S_{n_S}}\left(\mathbf{R} \underset{\theta}{\bowtie} \mathbf{S}\right)\right)\right) \tag{2.9}$$

[25, 72].

### 2.1.5 Algorithms

In this section, we describe algorithms commonly used by database management systems for binary operations within the database, set operations and joins in particular. These operations require identifying whether two records match on some subset of the field values, and there are two widely used approaches of accomplishing this: sorting and hashing. Once it is determined whether the rows match or not, the combined rows are either added to the result set or discarded, depending on the particular operation.

To sort relations too large to store in main memory, the data is partitioned into subsets that will fit into memory, and each of these subsets is sorted and stored. These stored, sorted partitions of the data are then merged, resulting in a fully sorted relation [25, 72].

Database hashing algorithms utilize two steps: a partitioning step and a probing step. The partitioning step of the basic hash algorithm builds a hash from the smaller of the two relations using a hash function *h*, or partitions it. The second relation is then scanned, hashed, and the original hash is then probed to determine if there is a matching record. In cases where the hash of the smaller table will not fit in main memory, a second hash function, $h_p$, is used to partition both relations, ideally into subsets of approximately uniform size. Using the same hash function to partition both relations means that matching records in the two relations are found in corresponding partitions, so the probing step consists of iterating through the partitions, probing for matching hashes using $h$. This requires two passes over the table, but maintains the $O(1)$ time complexity of the basic hashing algorithm [25, 72].

### 2.1.6 Concurrency Control

In many cases, it is desirable to permit concurrent updates to a database; however, if not handled correctly, problems can arise. The main problem is maintaining consistency; and this problem is addressed using the concept of transactions in conjunction with table locking.

Conceptually, transactions can be thought of as a group of database operations—select, insert, update, delete—that perform a single logical operation. For example, suppose we want to move a record from one table to a another: this transaction would consist of inserting the record into the target table and removing it from the source table. If either of these operations fails, the goal is not achieved.

Each transaction must maintain the consistency of the database. Since database consistency is, to a certain extent, application independent, the DBMS cannot be responsible for this, *i.e.* it is the responsibility of the application developer. If all transactions maintain the consistency of the database, the DBMS can ensure consistency by ensuring that each transaction is atomic: either every action in the transaction is completed or no action is completed. After all components of a transaction are executed, the transaction is finalized with a “commit” operation. Once the transaction has been committed, it must persist in storage: a property called durability. Finally, each transaction must execute in isolation; that is, it cannot be affected by any other transaction that is running concurrently. [72]

Isolation of transactions is directly related to the issue of schedule serializability. Suppose there is a set of transactions that are run concurrently. This set of transactions is said to be serializable if the result of the concurrent execution is guaranteed to be equivalent to the result of one of the possible serial execution schedules for the set of transactions, without any guarantee of the result matching a particular serial schedule. [72].

There are three issues that can arise when concurrent execution is allows without ensuring serializability:

- Dirty read: uncommitted data from another transaction is read.
- Non-repeatable read: a commit by another transaction results in consecutive reads of the same data item producing different values.
- Phantom read: a commit by another transaction results in two consecutive executions of the same query to produce different set of rows [87].

Two phase locking can be used to guarantee serializability. In this scheme, each data item can be locked in one of two ways: 1) a shared lock is obtained to read a value and allows other transactions to also obtain shared locks and 2) an exclusive lock must be acquired to write an object and can only be obtained when the data item is not locked by any other transaction. Once an exclusive lock is held, no additional locks, shared or exclusive can be acquired on the data item. During the first phase of the two phase locking protocol, all needed locks are obtained, no locks are released, and no lock is downgraded from exclusive to shared. In the second phase, all locks are released, no locks are obtained, and no locks are upgraded from shared to exclusive [72].

An alternative approach, and the approach relevant to the work presented in this dissertation, is multiversion concurrency control. In this case, some history of values for each data item is maintained. Each transaction, upon initiation, is labelled with a timestamp that indicates when execution began. When a value is used, the most recent value of the data item that was committed before the transaction began is read. Write operations are blocked if there is a shared lock on the data item held by a transaction started more recently [72].

Some database management systems use multiversion concurrency control but do not ensure serializability by default, *e.g.* PostgreSQL. Serializability is the highest of four isolation levels, and for each increase in isolation level, one additional anomaly of concurrent access is prevented:

- Read Uncommitted: all anomalies are possible.
- Read Committed: prevents only dirty reads.
- Repeatable read: prevents dirty and unrepeatable reads.
- Serializable: prevents all anomalies.

By default, PostgreSQL uses the read committed isolation level, which, in general, is not a problem for our application; however, it does raise one data loading issue which we describe in Chapter 3 [87].

## 2.2 Data Warehousing

Data warehouses serve a fundamentally different function than databases. Databases are typically used to store operational data that must remain up to date; data warehouse store historical data.

### 2.2.1 Warehouse Design

In his canonical work on data warehousing, Inmon [45] describes several important elements of warehouse design, including granularity, partitioning, and normalization.

According to Inmon, the most important of these is the granularity of the data. The granularity refers to the level of detail of the stored data; and Inmon provides a very concise explanation regarding the importance of choosing the correct granularity:

> Granularity is the single most critical design issue in the data warehouse environment because it profoundly affects the volume of data that resides in the data warehouse and the type of query that can be answered [45].

Data with higher granularity can be store in less space; however, fewer types of results can be obtained from the data [45].

Another significant aspect of warehouse design is data partitioning. Without partitioning, certain tables in the warehouse may become so large that they are no longer manageable with respect to indexing, sequential scans, and data reorganization. There are two approaches to data partitioning: system level and application level. In system level partitioning, the partitioning of the data is handled by the database management system. In application level partitioning, the partitioning is handled by the applications developed for manipulating the warehouse and in this case the partitioning scheme is not represented in the database management system in any way. Generally, the latter is preferred because it is the more flexible approach. The main drawback of the system level partitioning is that, often, only a single data definition is permitted for the partitions of a particular relation [45]. For example, in PostgreSQL table partitioning is accomplished by creating a root table that will contain no rows and creating a set of tables that inherit the fields of this table. Querying the root table accesses the rows in the children. Since all tables in the partitioning for a single logical table represented by the parent, they must have the same set of fields. The elements of the partition can, however, be indexed independently. Additionally, each element of the partition can be defined with field constraints to ensure that each row is inserted into the proper partition. These constraints can also be leveraged by the query planner to exclude scans of partition elements that do not contain any relevant records [87].

The final aspect of warehouse design that we will discuss is that of data normalization vs. denormalization. The trade-off to consider in this case is that of storage requirements and I/O cost. The issues of update and delete anomalies are not relevant since, once populated, the warehouse is a read-only data repository. Insertion anomalies can be handled by the data loading applications. Data normalization reduces the redundancy in the data and, therefore, reduces the volume of storage required for the data. The cost of the reduction in storage requirements comes when the data must be retrieved: the records must be re-assembled from various tables, and accessing each table incur I/O costs. In some cases, two levels of normalization are used to reduce both storage and I/O costs. Data that is accessed often remains in a less normalized form and data that is rarely accessed is stored in a more normalized form [45].

# CHAPTER 3

# DESIGNING A DATA WAREHOUSE FOR SOCIAL NETWORK AND HUMAN MOBILITY RESEARCH

## 3.1 Introduction

In this chapter, we describe a data warehouse for complex network and human mobility research. The warehouse currently stores 14 months of call data from a cellular service provider and continues to grow.

Motivated by difficulties in dealing with the flat files provided by the service provider and the fact that researchers often perform the same set of cleaning steps before using the data, we move the data into a database management system, do general cleaning of the data, and organize the records using a star schema and table partitioning. Our goal is to provide a repository for efficient extraction of subsets for the data for specific research projects through simple SQL queries.

This chapter is organized as follows: section 3.2 discusses related work in data organization, section 3.3 describes the design of the warehouse, section 3.4 describes the implementation of the warehouse, including a description of the flat files from which the data is obtained (section 3.4.1), the data validation and loading process (section 3.4.2, and merging and partitioning of the usage table (sections 3.4.5 and 3.4.6, respectively).

## 3.2 Related Work

Traditionally, researchers in the sciences have relied on standardized file formats for organizing data and achieving data independence. For example, the Flexible Image Transport System (FITS) is a data format developed for sharing images from telescopes. The format consists of a header that contains the information about the data, the meta-data, required to effectively use the images along with the data itself [92]. The NetCDF file format was developed for a similar purpose in the atmospheric sciences community. NetCDF is an extension of NASA's Common Data Format and utilizes Sun's eXternal Data Representation to store the data in a machine independent manner [73]. The Hierarchical Data Format (HDF) is a library for sharing technical and scientific data using a "self-describing" file format [29].

A major drawback of these early file formats is the lack of a powerful search function. As the volume of available data increases, the existing functionality to find the data items of interest becomes prohibitively expensive [36].

The lack of adequate search functionality also arises in bioinformatic databases; the INTERACT database for the study of protein interactions was developed in response to this issue as well as the problem that protein interaction is spread over a number of distinct databases [24]. Development of the INTERACT database was motivated by the difficulty in extracting protein interaction data from the yeast protein database (YPD) [43] and the yeast genome database of the Munich Information Center for Protein Sequences (MIPS) [61]. The Yeast Protein Database (YPD) contains manually curated documents about each protein found in *Saccharomyces cerevisiae*, and it is in these documents that the interaction data is found [43]. The Munich Information Center for Protein Sequences (MIPS) yeast genome database also contains information about the proteins found in *Saccharomyces cerevisiae*, but is organized by gene name [61]. The

INTERACT database contains the data from the MIPS database along with data provided by scientists and data from the literature organized in such a way that researchers studying protein interactions can readily obtain relevant data. The design was developed in consultation with potential users.

In the bioinformatics community, the data has become spread across a large number of databases. It is often desirable to access data distributed across multiple locations; consequently, three techniques have arisen to accomplish this: 1) link driven federation, 2) view integration, and 3) data warehousing. Link driven federation consist of links in the results of one data source that direct users to additional information. View integration provides a query system that, in turn, queries a set of remote sites, receives and processes the results, and presents an integrated result to the user. A data warehouse stores the data from the remote sites at a central location [21]. More recently, several additional warehousing systems have been developed, including EnsMart [48], BioWarehouse [53], and the Biochemical Network Database (BNDB) [52].

Additionally, very large databases are found in the areas of astronomy and high energy physics. As of 2005 the Sloan Digital Sky Survey (SDSS) had approximately 50 TB of raw image data. From this data, 1.6 TB of catalog data derived from the raw images, organized, and stored in an database and made available online through SkyServer via a number of query mechanisms [1, 62]. The storage requirements for high energy physics research are even greater. As of 2005, the BaBar database, developed at CERN, contained approximately 10 billion data items and required 1.3 PB of storage. The data is accessed through the metadata, which is heavily indexed to improve access efficiency [6].

The system we describe in this chapter merges data from multiple sources in a single company and organizes the data to allow efficient access.

## 3.3 Warehouse Design

In this section, we describe the design of the warehouse. We describe the data set and the database schema used to organize the data in the database management system. Additionally, we describe our data partitioning approach that, in many cases, prunes irrelevant rows from the table scans.

### 3.3.1 Data Overview

Our warehouse consists of three major tables: the usage table describes service usage on the network, the customer table which describes the customers of the service provider, and the antenna table which provides location information associated with the towers owned by the service provider. Note that all user identifiers have been de-identified by the service provider. Tables 3.1, 3.2, and 3.3 describe the fields of the usage, customer, and antenna tables, respectively.

Each record in the usage table can be uniquely identified by the time at which the service is activated, the de-identified account initiating the service, the de-identified account receiving the service, and the type of service being accessed; therefore, of the fields in the usage table, `start time`, `call from`, `call to`, `service` form the primary key. The `call from` and `call to` fields can be linked to additional data when they contain values corresponding to current individual customers of the company. The `antenna from` and `antenna to` fields can be linked to geographical information found in the `antenna` table.

For simplicity, we limit the discussion of the customer data to the fields that are relevant to the design; specifically, the fields that allow us to identify active accounts

TABLE 3.1

DESCRIPTION OF THE FIELDS IN THE USAGE TABLE

| **Field name** | **Description** |
|---|---|
| `start time` | The time the service is activated |
| `call from` | The account initiating the service |
| `call to` | The account receiving the service |
| `service` | The type of service being accessed, *e.g.* call, SMS |
| `destination code` | A company generated code providing additional information about the recipient |
| `duration` | The length of time the service is in use |
| `cost` | The amount charged to the account for the usage instance |
| `antenna from` | The antenna the initiating device is connected to |
| `antenna to` | The antenna the receiving device is connected to |

of the service provider in the usage data: the activation date, the day the account was opened, the disconnect date, the day the account was closed or null if the account is still open, and the service type, which indicates whether the account is prepaid or postpaid. We use these fields to partition the usage data based on whether the caller and recipient have active accounts with the service provider.

Similarly, we link the antenna fields in the usage data to a table that gives location information for the company's towers. Each antenna is uniquely identified by an integer ID. For many of these antennas we have the location of the tower (latitude and longitude). We also have GIS data describing various features of the geographical area covered by the phone data that we include in tables to facilitate spatial queries. The most prominent of these features is a set of shape files that describe the postal

TABLE 3.2

DESCRIPTION OF THE FIELDS IN THE CUSTOMER TABLE

| **Field Name** | **Description** |
|---|---|
| `effective date` | The month in which the record for the account was last updated. |
| `phone id` | The de-identified phone number identifying the account. |
| `activation date` | The date on which the account was connected. |
| `disconnect date` | The date on which the account was terminated. This field is `NULL` for accounts that are active on the effective date. |
| `service type` | The type of payment plan for the account: prepaid or postpaid. |

codes. From these values, we add additional fields to the antenna table: the `tower` field uniquely identifies each distinct antenna location, the `tower point` is a shape object that contains the geographic point of the tower, the `postal code` and `postal code` points identify the postal code containing the tower, and the city identifies the city in which the tower is located.

The customer and antenna tables are relatively small; however, the usage table contains at least 1 billion records per month. In the next section, we describe how the data is partitioned into subsets of manageable size.

### 3.3.2 Usage Table Partitioning

We partition the usage table for two reasons: 1) there are well known performance benefits to partitioning the fact table of a data warehouse [44], and 2) there are subsets of the data that the researchers are likely to want that can be pre-partitioned to eliminate

TABLE 3.3

DESCRIPTION OF THE FIELDS IN THE ANTENNA TABLE

| **Field Name** | **Description** |
|---|---|
| `antenna` | An integer uniquely identifying the antenna. |
| `tower` | An integer identifying the tower on which the antenna is mounted. |
| `latitude` | The latitude of the tower. |
| `longitude` | The longitude of the tower. |
| `tower point` | A geometry field containing the point at which the tower is located. |
| `tower voronoi cell` | A geometry field containing the boundary of the cell of a Voronoi diagram containing the tower. |
| `postal code` | The postal code containing the tower. |
| `postal code shape` | A geometry field containing the boundary of the postal code containing the tower. |
| `city` | The name of the city in which the tower is located. |

the need to perform sequential scans or joins over the entire set of usage records.

We look to past publications using the dataset and conversations with the researchers to develop the partition hierarchy. In practice, the data is often filtered along two axes: 1) service type and 2) the relationship of the accounts with the service provider.

In most cases, the only services of interest are voice calls and SMS: Onnela *et al.* [63] use only voice calls in their study of graph robustness with respect to the removal of ties of varying strength. González *et al.* [34] and Wang *et al.* [91] use only voice calls SMS in their studies of human mobility and the spreading of viruses to mobile phones, respectively. We, therefore, partition the records into three tables based on

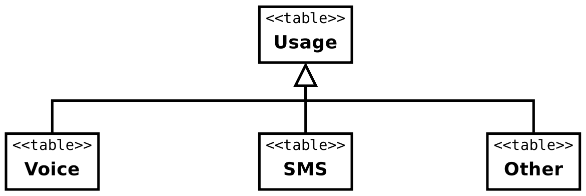


Figure 3.1. The service hierarchy.

service: one table contains only voice call records, a second table contains only SMS records, and a third table contains the records of all other services. Figure 3.1 illustrates this portion of the hierarchy.

Additionally, it is often desirable to partition the records based on whether the caller and recipient are current customers of the company because the data is most complete for this set of accounts. Onnela *et al.* [63] use only records where both the caller and recipient are current customers of the company so that they have the complete call activity for each node in the network. In human mobility studies, it may be sufficient to restrict only the caller to current customers of the company and impose no constraint on the recipient, depending on the role of the social network in the study.

We partition the usage data into 4 groups based on whether the caller and recipient are current customers of the company based on the data available in the customer table:

- *in-in*: both the caller and recipient are current
- *in-out*: only the caller is a current customer
- *out-in*: only the recipient is a current customer
- *out-out*: neither the caller nor the recipient are current customers

We also partition the usage records based on whether "in-network" users have prepaid or postpaid accounts. Figure 3.2 illustrates this portion of the hierarchy for the voice call service. Identical hierarchies exist for the “SMS” and “other” branches of the usage hierarchy.

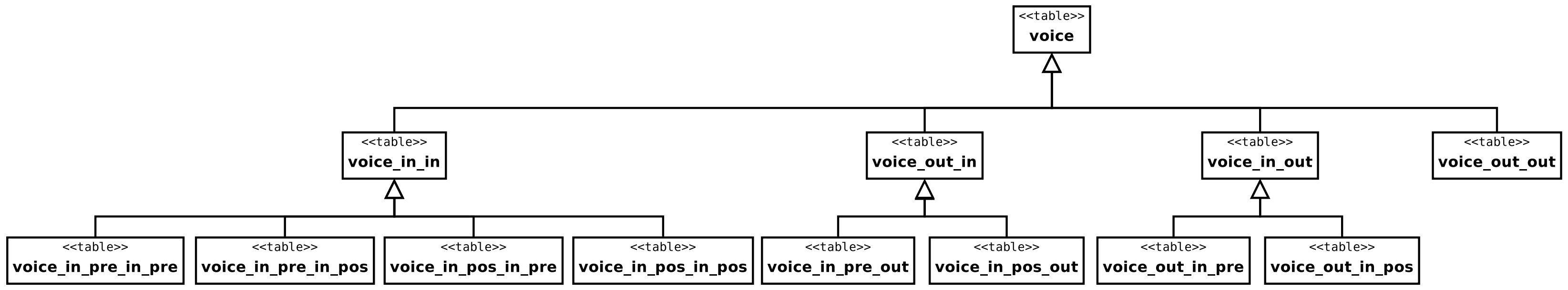


Figure 3.2. The customer status hierarchy for the voice (phone call) service. The top level is an abstract table that, when queried, accesses all voice records. The second level partitions the records based on whether the caller and recipient are current customers of the service provider. The bottom level further partitions the records of current customers based on whether they are prepaid or postpaid customers. Identical hierarchies exist for the SMS and other components.

### 3.3.3 Schema

We use a variant of the star schema for the warehouse. The star schema consists of two components: the fact table, which contains a history of measurements taken over a period of time, and the dimensional tables, which contain descriptive information about elements of the fact table. The elements in the fact table are related to records in the dimensional table using foreign key constraints [50]. In our warehouse, the usage table is the fact table and the customer and antenna tables are dimensional tables. Figure 3.3 shows the tables organized in a standard star schema.

Unfortunately, the dimensional model is problematic for our warehouse because the foreign key constraint requires that for each value in a field associated with a dimensional table, *e.g.* `call_from`, there must be a corresponding record in the dimensional table. Our records include calls to and from phones owned by customers of other companies, so we do not have customer records for most of the values that appear in the `call_from` and `call_to` fields of the usage table due to the market share of the service provider. One possible solution is to pad the customer table with empty records; however, this would more than double the size of the table without adding any new data.

Alternatively, we can leverage the fact that the usage table has been partitioned based on whether the `call_from` and `call_to` fields are in the customer table. For each partition, we add only the appropriate foreign key constraints. Figure 3.4 shows the modified star schema.

## 3.4 Warehouse Implementation

To build the warehouse, we validate the fields of each row, load the data into temporary tables, and use the database management system to transform the data into its

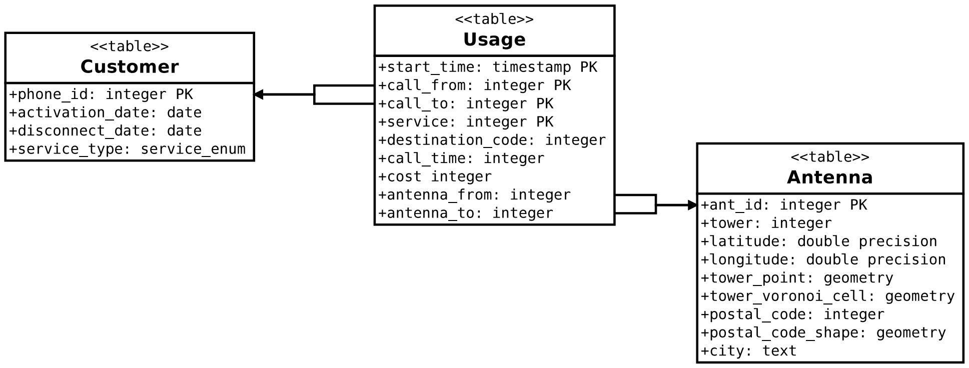


Figure 3.3. The standard star schema for the warehouse. The usage table records the facts of the warehouse and references the customer to provide further information about the subscribes and the antenna table to provide additional location information.

final form. In this section, we describe these processes.

### 3.4.1 Data Sources

A company provides the data using a variety of file types. Three of these file types contain the usage data: call, interconnect, and call record data (CDR). The call and interconnect files provide billing information, such as the duration and cost, of each use of a service. The CDR files contain location information for each call and SMS. The customer data arrives after each month in 4 files containing current prepaid accounts, current postpaid accounts, prepaid accounts that have been deactivated during the month, and postpaid accounts that have been deactivated during the month. In addition to these data sets, we have location data for the antennas that are not updated.

Figure 3.5 shows how the source files are merged into the usage and customer tables. Details of this process are provided in the next section.

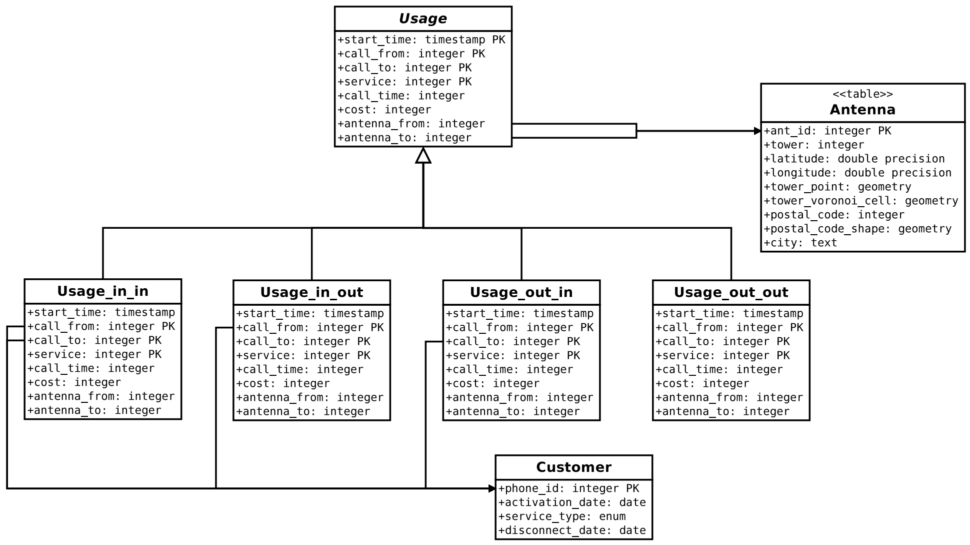


Figure 3.4. The partitioned fact table schema. The fact table is partitioned based on whether the caller and recipient are customers of the company and foreign keys are only applied in cases where there is a corresponding record in the customer table, *i.e.* in cases where the caller or recipient is a current customer.

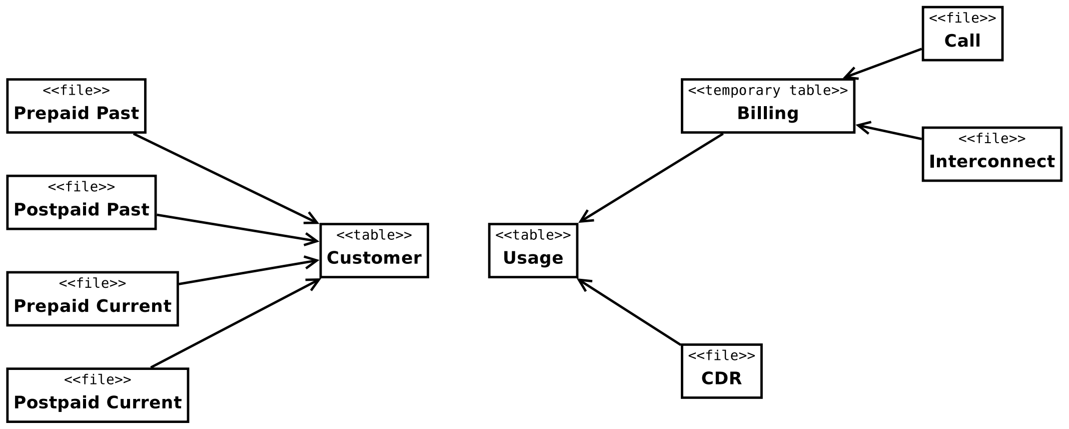


Figure 3.5. Overview of the data source files

### 3.4.2 Data Loading

The data loading process consists of three steps: 1) record validation, 2) record insertion, and 3) user string replacement.

Validating and transferring the data from the file server to the database server is very time consuming; however, we are able to accelerate the process via parallelization. Each record is validated using a perl script which first splits each record into an array and then uses regular expressions to ensure that the fields are in the required form. This allows us to catch erroneous records that would cause the bulk load to terminate and set them aside for further cleaning. Using a single thread we are able to validate and transfer 4 GB / hour of data from the file server to the database. The machine being used for validation has 8 cores, so rather than serially validating all of the files for a month, we validate them concurrently, loading them into temporary tables, and finally merging these tables. With this approach, we've achieved validation and transfer rates of up to 27 GB / hour using 7 threads.

Once the data is in the database, we replace the hash strings, the de-identified user identifiers provided by the company, with integers, eliminate duplicate rows that appear in the raw data, and separate rows with non-unique primary keys for further analysis.

#### 3.4.2.1 Hash Replacement

Before releasing the data, the service provider de-identifies the phone numbers using a hash function. To reduce storage requirements and the time required for comparison operations, we replace the 27 character strings with a 4 byte integers. We keep a table, a hash map, that contains a (hash, integer) pair for each hash value we encounter in loading the data. For each file we load, we identify the new hashes, update the hash map, and replace each hash value with the corresponding integer.

In this section, we first define the relation that maps hashes to integers; we then describe our approaches for updating this map and replacing the hashes in the warehouse tables, including proofs of correctness where necessary.

Let $\mathbf{H} = (H, I)$ be the hash map relation where $I$ is the integer representing hash $H$ such that $\mathbf{H}$ is a function $\mathbf{H} : H \longmapsto I$. By definition, each $H$ maps to a unique $I$. This constraint ensures that the hashes will be replaced in a consistent manner. We leverage the DBMS to enforce this constraint using a table where the hash field is the primary key, requiring each row to have a unique hash value, and the ID field is an auto-incrementing integer. Table 3.4 shows the PostgreSQL definition of this table.

For each file we process, we need to add previously unseen hashes to the hash map. To accomplish this, we insert the items in the set difference between the hashes in the usage and hash map tables:

$$\pi_{H_u}(\sigma(\mathbf{U})) - \pi_H(\sigma(\mathbf{H})) \tag{3.1}$$

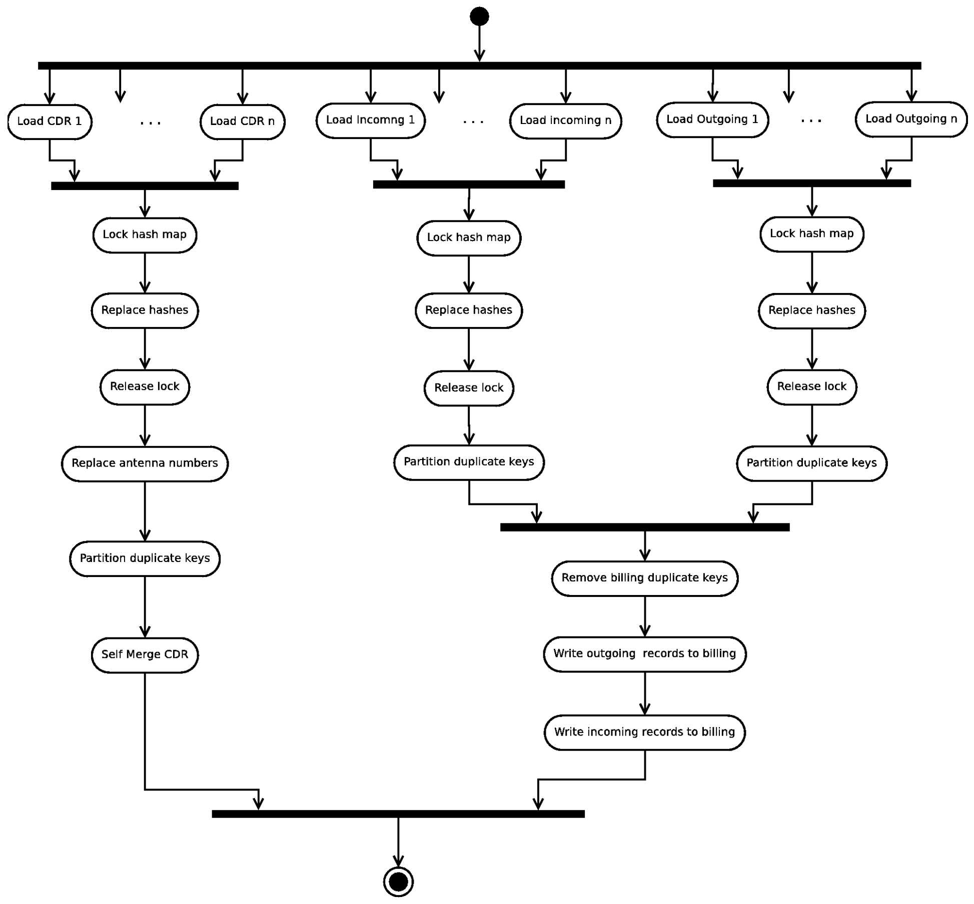


Figure 3.6. Activity diagram for validating the data contained the flat files and loading these into the DBMS. Most of the process can be parallelized, with the exception of the hash replacement. The PostgreSQL concurrency mechanism does not sufficiently handle our case, so we must use explicit table locks to insure that the constraints of the hash map are not violated.

TABLE 3.4

THE HASH MAP TABLE

| **Column** | **Type** | **Modifiers** |
|---|---|---|
| `hash` | `text` | `PRIMARY KEY` |
| `id` | `integer` | `NOT NULL default nextval('hash_map_id_seq'::regclass)` |

The PostgreSQL query evaluation mechanism computes this set difference via sorting, requiring $O((|\mathbf{U}| + |\mathbf{H}|) \log (|\mathbf{U}| + |\mathbf{H}|))$ time. Alternatively, we can obtain the set difference using a left join,

$$\pi_{H_u}\Bigg(\sigma_{I=\omega}\Bigg(\sigma_{H_u=H}(\mathbf{U} \times \mathbf{H}) \cup \Big((U - \pi_{U.*}(\sigma_{H_u=H}(\mathbf{U} \times \mathbf{H}))) \times (\omega, \omega)\Big)\Bigg)\Bigg) \tag{3.2}$$

Since PostgreSQL uses hash joins, this query computes the set difference in $O(|\mathbf{U}|+|\mathbf{H}|)$ time.

We now show that the left join query in equation 3.2 is equivalent to the set difference query in equation 3.1. First, we establish that in relational algebra selection is distributive over set unions. We then use this result to show the equivalence.

**Lemma 1.** *Selection* ($\sigma$) *is distributive over union* ($\cup$), i.e.

$$\sigma_\theta(\mathbf{R}_1 \cup \mathbf{R}_2) \equiv \sigma_\theta(\mathbf{R}_1) \cup \sigma_\theta(\mathbf{R}_2) \tag{3.3}$$

*Proof.* To prove this equivalence, we show that for an element $r$,

1. $r \in \sigma_\theta(\mathbf{R}_1 \cup \mathbf{R}_2) \implies r \in \sigma_\theta(\mathbf{R}_1) \cup \sigma_\theta(\mathbf{R}_2)$,

2. $r \notin \sigma_\theta(\mathbf{R}_1 \cup \mathbf{R}_2) \implies r \notin \sigma_\theta(\mathbf{R}_1) \cup \sigma_\theta(\mathbf{R}_2)$,

3. $r \in \sigma(\mathbf{R}_1 \cup \mathbf{R}_2) \implies r \in \sigma_\theta(\mathbf{R}_1) \cup \sigma_\theta(\mathbf{R}_2)$, and

4. $r \notin \sigma(\mathbf{R}_1 \cup \mathbf{R}_2) \implies r \notin \sigma_\theta(\mathbf{R}_1) \cup \sigma_\theta(\mathbf{R}_2)$.

Suppose a row $r \in \sigma_\theta(\mathbf{R}_1 \cup \mathbf{R}_2)$. Then $r$ satisfies $\theta$ and $r \in \mathbf{R}_1$ or $r \in \mathbf{R}_2$. Therefore, $r \in \sigma_\theta(\mathbf{R}_1) \cup \sigma_\theta(\mathbf{R}_2)$. Now suppose that a row $r \notin \sigma_\theta(\mathbf{R}_1 \cup \mathbf{R}_2)$. There are two cases to consider: 1) $r$ does not satisfy $\theta$, in which case $r \notin \sigma_\theta(\mathbf{R}_1)$ and $r \notin \sigma_\theta$ and 2) $r \notin \mathbf{R}_1 \cup \mathbf{R}_2$, in which case $r \notin \mathbf{R}_1$ and $r \notin \mathbf{R}_2$. In either case, $r \notin \sigma_\theta(\mathbf{R}_1) \cup \sigma_\theta(\mathbf{R}_2)$. Thus,

$$r \in \sigma(\mathbf{R}_1 \cup \mathbf{R}_2) \implies r \in \sigma_\theta(\mathbf{R}_1) \cup \sigma_\theta(\mathbf{R}_2) \tag{3.4}$$

and

$$r \notin \sigma(\mathbf{R}_1 \cup \mathbf{R}_2) \implies r \notin \sigma_\theta(\mathbf{R}_1) \cup \sigma_\theta(\mathbf{R}_2). \tag{3.5}$$

Now suppose row $r \in \sigma_\theta(\mathbf{R}_1) \cup \sigma_\theta(\mathbf{R}_2)$. Then $r$ satisfies $\theta$ and $r \in \mathbf{R}_1$ or $r \in \mathbf{R}_2$. Thus, $r \in \sigma_\theta(\mathbf{R}_1 \cup \mathbf{R}_2)$. Now suppose that $r \notin \sigma_\theta(\mathbf{R}_1) \cup \sigma_\theta(\mathbf{R}_2)$. There are two cases to consider: 1) $r$ does not satisfy $\theta$, in which case $r \notin \sigma_\theta(\mathbf{R}_1 \cup \mathbf{R}_2)$ and 2) $r \notin \mathbf{R}_1$ and $r \notin \mathbf{R}_2$, which case $r \notin \mathbf{R}_1 \cup \mathbf{R}_2$. In either case, $r \notin \sigma_\theta(\mathbf{R}_1 \cup \mathbf{R}_2)$. Thus

$$r \in \sigma_\theta(\mathbf{R}_1) \cup \sigma_\theta(\mathbf{R}_2) \implies r \in \sigma_\theta(\mathbf{R}_1 \cup \mathbf{R}_2) \tag{3.6}$$

and

$$r \notin \sigma_\theta(\mathbf{R}_1) \cup \sigma_\theta(\mathbf{R}_2) \implies r \notin (\mathbf{R}_1 \cup \mathbf{R}_2). \tag{3.7}$$

Since expressions 3.4, 3.5, 3.6, and 3.7 are true, we know that $\sigma_\theta(\mathbf{R}_1 \cup \mathbf{R}_2) \equiv \sigma_\theta(\mathbf{R}_1) \cup \sigma_\theta(\mathbf{R}_2)$. □

**Theorem 1.** *Let* $\mathbf{U} = (H_u)$ *and* $\mathbf{H} = (H, I)$ *be relations. The set difference*

$$\pi_{H_u}(\mathbf{U}) - \pi_H(\mathbf{H}) \tag{3.8}$$

*is equivalent to the left join*

$$\pi_{H_u}\Big(\sigma_{I=\omega}\Big(\sigma_{H_u=H}(\mathbf{U}\times\mathbf{H})\cup\Big((U-\pi_{U.*}\,(\sigma_{H_u=H}(\mathbf{U}\times\mathbf{H})))\times(\omega,\omega)\Big)\Big)\Big) \tag{3.9}$$

*Proof.* To prove this equivalence, we start with the expression for the left join and derive the set difference expression. First, we distribute the selection on $I=\omega$:

$$\sigma_{I=\omega}\Big(\sigma_{H_u=H}\Big(\mathbf{U}\times\mathbf{H}\Big)\cup\Big((\mathbf{U}-\pi_{U.*}\,(\mathbf{U}\times\mathbf{H}))\times(\omega,\omega)\Big)\Big) \tag{3.10}$$

Since none of $(H,I)\in\mathbf{H}$ and $H_u\in\mathbf{U}$ can be `NULL` ($H$ and $H_u$ are at least part of the primary key and there is an explicit constraint that prevents $I$ from being `NULL`), $\sigma_{I=\omega}(\sigma_{H_u=H}(U\times H))=\emptyset$. Thus, we can drop the left hand side of the union to get

$$\pi_{H_u}\Big(\sigma_{I=\omega}\Big((\mathbf{U}-\pi_{H_u}\,(\sigma_{U.*}(\mathbf{U}\times\mathbf{H})))\times(\omega,\omega)\Big)\Big) \tag{3.11}$$

This expression gives the set difference between the values of $H_u\in\mathbf{U}$ and the values of $H_u$ that are also in the field $H\in\mathbf{H}$. Therefore, if a $h$ is in the result set of the above query, then $h\in\pi_{H_u}(\mathbf{U})$ and $h\notin\pi_{H_u}(\mathbf{U})\cap\pi_H(\mathbf{H})$, thus $h\in\pi_{H_u}(\mathbf{U})-(\pi_{H_u}(\mathbf{U})\cap\pi_H(\mathbf{H}))\equiv\pi_{H_u}(\mathbf{U})-\pi_H(\mathbf{H})$. □

PostgreSQL uses, by default, the read committed isolation level, so there is nothing to prevent phantom reads, the case where a commit by one transaction changes the set of rows that would be produced by a select statement in another transaction. A problem arises when a new hash value appears in two concurrent insertions: the second insertion of the value, which is based on the result of a select statement that is no longer valid, will violate a uniqueness constraint on the hash field required to ensure that the hashes map consistently to a single integer. Since PostgreSQL does not block the second hash map update query, we must explicitly obtain an exclusive lock on the entire hash map table when updating it. Figure 3.6 shows the activity diagram for validating and loading the data in parallel, including the hash map table locks.

To replace the hashes with integers, we use an inner join. Since each unique hash maps to a unique integer, we replace the hashes using an inner join,

$$\pi_I\left(\mathbf{U} \underset{H_u=H}{\bowtie} \mathbf{H}\right), \tag{3.12}$$

without introducing duplicate records.

Once the hashes are replaced, we proceed with merging the data into unified tables.

### 3.4.3 The Customer Table

Each month of customer data is loaded from four files. Two files, prepaid and postpaid, identify accounts that have been closed during the month. Two similar files identify active accounts. There is significant overlap among the fields in these files, so we merge them into a single table.

Only a small number of the records for the current users change from month to month; therefore, we considered using a single customer table with an additional field indicating the effective date of the record. In this case, we only add records to the customer data when at least one field for a customer has changed, including account termination or activation. To retrieve the churn data for a given month, $m$, we select the record for each customer $i$ with the latest effective date $e$ such that $e \leq m$. Let $\mathbf{C} = (E, I, \ldots)$ be our customer relation. The query for selecting the customer data for month $m$ is

$$\sigma_{(E,I)\in\pi_{\max E,I}(\sigma_{E\leq M}(\mathbf{C}))}(\mathbf{C}) \tag{3.13}$$

This organization results in a tradeoff: we reduce the storage requirements; however, queries on this data are more computationally expensive due to the join required to extract a particular month of data.

Beyond the issue of the time-storage trade-off, this organization only works if the data is clean, which it is not in this case. The problem is that any missing or spurious records propagate to later months. We discovered that, due to this propagation, the data yielded several million additional users for the later months than there should have been[1].

### 3.4.4 The Antenna Table

The antenna data was compiled, cleaned, and loaded into a PostgreSQL database by Schoenharl [77]. The antenna data initially consisted of an antenna ID, the latitude and longitude of the tower on which the antenna is mounted, and the postal code in which the tower is located.

We use PostGIS, an extension of PostgreSQL that implements the OpenGIS standard, to represent spatial objects in the warehouse. PostGIS provides a geometry field type that stores spatial objects, such as points and polygons. It also provides efficient spatial functions that can be included in SQL queries such as `contains`, a boolean function that returns true if one spatial object is completely within the bounds of another, and `touches`, a boolean function that returns true if two spatial objects tangentially intersect.

We added some additional fields to this table from other sources, including a geometry object giving the borders of the postal codes (also from a table built by Schoenharl [77]), a geometry field containing a point giving the location of the tower, and the shape of the Voronoi cell in which the tower is located (this Voronoi diagram has been used

[1]The spurious users were discovered by Dr. Jim Bagrow and Dashun Wang.

for a variety of purposes in [34, 77]).

### 3.4.5 The Usage Table

The call, interconnect, and CDR tables are combined to form the usage table. The call and the interconnect call files contain the duration and cost of each call; therefore, we refer to these as billing data. The CDR files contain the antennas that each call is routed through and provide coarse location data. The primary key of the usage files consists of the start time of the call, the ID of source phone, the ID of the receiving phone, and the service used. Since the start time of the call is measured to the second, we assume that these fields form a unique identifier for each record. We, therefore, use this primary key to match the call records among the three file types.

#### 3.4.5.1 The Billing Table

The billing data consists of two data sets that are, in theory, disjoint. The call data consists of records of service usage by customers of the company and the interconnect data contains calls and SMS received by customers of the company from people who are not customers.

We find a small number of cases where calls are recorded in both the call and interconnect data. This appears to be an artifact of the way in which the customer's status is determined during the generation of these files, and we describe how we deal with this situation in the next section. For the time being we include the record from the call file in the billing table and put the record from the interconnect file into a separate location.

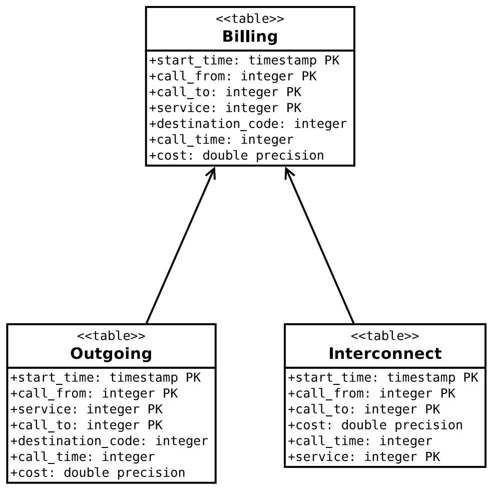


Figure 3.7. Merging the call and interconnect tables to produce the billing table.

#### 3.4.5.2 The CDR Table

The CDR data contains the antennas through which each voice call and SMS is routed. There are four types of records in the CDR data: originating and terminating records for voice calls and SMS messages. The MOC and MTC records give the tower information for the caller and the recipient, respectively, for a voice call. Similarly, the SOM and STM records give the tower information for the sender and recipient, respectively, of an SMS message.

The MOC and MTC records are merged using a full outer join, matching records on `start time`, `call from`, and `call to` (all these records describe usage of the voice service).

The SOM records are generated at the time the SMS is sent and the STM records are generated at the time the SMS is downloaded by the recipient. Therefore, the SOM and STM records cannot be merged into a single record. The SOM records are inserted into the CDR table with the antenna number in the `antenna from` field and a `NULL` value in the `antenna to` field. The STM records are inserted into the CDR table with `NULL` value in the `antenna from` field and the antenna number in the `antenna to` field.

### 3.4.6 Data Partitioning

We leverage the table inheritance functionality of PostgreSQL to implement the partitioning of the usage table. Only the leaf tables in the hierarchy contain records; the tables at higher levels are formed by concatenating a subset of the leaves. The tables at the service level of the hierarchy are constrained by service. With these constraints, the query planner can prune irrelevant service subtrees (voice, SMS, and other) based on the service values specified in the `WHERE` clause of a SQL statement. Similarly, the `start time` field in each leaf table is constrained to only allow records from a particular month, allowing the query mechanism to omit tables outside the `start time` range specified in the query from the database search.

Separating the records based on service is straightforward and can be accomplished by examining the service field; however, partitioning the records based on whether the caller or recipient is a customer of the company is more complicated. We merge the usage and customer tables using a left join on the `call from` field and again on the `call to` field:

$$(\mathbf{U} \underset{\mathbf{U}_{C_f}=\mathbf{C}_C}{\ltimes} \mathbf{C}) \underset{\mathbf{U}_{C_t}=\mathbf{C}_C}{\ltimes} \mathbf{C} \tag{3.14}$$

This produces a table that gives the activation and disconnect dates for the caller and recipient in each row of the usage table. We then process each row in the resulting table and place it into the correct partition based on the `service` field and the following rules for determining whether the account is part of the company's network or another network:

- If the `start time` is after the `activation date` and it is either before the `disconnect date` or there is no `disconnect date`, the field corresponds to a current customer. We use the `service type` field to determine if it is a prepaid or a postpaid account.

- If the `start time` is before the `activation date`, after the `disconnect date`, or if there is no `activation date`, the field corresponds to a non-customer.

We also implement views to provide data in a format compatible with applications written for the original data files. There are simple rules that we can use to determine which file types—call, interconnect, and CDR—a record, or part of a record, appears in:

- Records from the call files have values in the destination code and cost fields.
- Records from the interconnect files have values in the cost field and are `NULL` in the destination code field.
- Records from the CDR files have a value in at least one of `antenna from` or `antenna to`.

Queries performed on these views will, like the usage table they are generated from, utilize the table constraints to prune unnecessary leafs from the set of table scans.

## 3.5 Summary and Conclusion

In this chapter, we have described the design of a large data warehouse for social network and human mobility research. We have leveraged the PostgreSQL database management system to facilitate data cleaning and organization. We sped the data loading process by showing we can use a left join to compute set difference in linear time rather than using the SQL set operations which require $O(n \lg n)$ time. Additionally, we parallelized the process where possible and used manual table locks where the PostgreSQL concurrency mechanisms were insufficient.

The design of the warehouse is based on actual use cases for the data and provides a substantial improvement over the use of flat files. The partitioning scheme, in many cases, allows many records irrelevant to particular research projects to be ignored altogether and yields a straightforward variant of the star schema that can handle the case where there are no dimensional records for most values in a field of the fact table.

It should be noted that only minimal data cleaning is performed as part of the data loading process, and the warehouse described only provides data storage. At this point, no analysis tools are provided. In the next chapter, we perform additional data cleaning to eliminate redundancy introduced when merging the elements of the usage table.

## 3.6 Acknowledgements

The author would like to thank Dr. Albert-Lázsló Barabási and the Center for Complex Network Research at Northeastern University for their support and input on the work described in this chapter. In particular, the author is grateful to Dr. Marta González, Dr. Julián Candia, Dr. Sune Lehmann, Dr. Jim Bagrow, Nick Blumm, and Dashun Wang for sharing their knowledge of the data and insights on how this data may be used for scientific research.

# CHAPTER 4

# AN APPROACH FOR MERGING NOISY DATA IN A SCIENTIFIC DATA WAREHOUSE

## 4.1 Abstract

In this chapter, we examine the consequences of naively merging the CDR and billing records as described in the previous chapter. We show that there is a time synchronization issue between the MOC and MTC records in the CDR data and, to a lesser extent, between the call and CDR data that introduces redundancy into the data set when merged. We describe and evaluate an approach for merging the data that corrects for the redundancy.

## 4.2 Introduction

When first merging the CDR data, we discovered that the number of calls for which we have both the MOC and the MTC records was significantly smaller than we expected (around 20%). There are several reasons why only one of the MOC or MTC record may be present. Suppose the caller or the recipient is either using a landline or outside the coverage area of the service provider. In this case, one of the records for that call is missing. We can account for some of this by including only records where the caller and the recipient are customers of the service provider. This eliminates all landlines and calls to customers of other service providers, who are likely using towers owned by that company but does not eliminate all cases where only one of the records is present (data will be missing for roaming users), we expect it to significantly improve the fraction of transaction for which we have both records. Unfortunately, we discover that the rate of matching is still fairly low (around 40%).

In this chapter, we show that the low matching rate is due to noise in the timestamp of the records and that merging the tables naively results in redundant records in the tables. We also present an approach for reducing the effect of this noise by merging MOC and MTC records that likely describe the same call. In section 3, we consider related work in the areas of record linkage, redundancy elimination, and event correlation. Section 4 describes our approach for merging the tables in the presence of noise. We conclude with the results of the merge in Section 5 and discussion in Section 6.

## 4.3 Related Work

Redundancy in data can arise in several different ways. It can be the result of merging databases from different sources into a single warehouse, or it can arise from multiple observations of the same event in a sensor network. Several approaches have been developed to address this problem, including record linkage, redundancy elimination, and event correlation.

Record linkage is an approach used in building data warehouses that matches records from different sources, which may have small errors in the key fields, using only evidence in the records themselves to determine whether two records represent the same entity.

Fellegi and Sunter [27] formalize a theory of record linkage where they assume that two sets of records are generated by two distinct processes. In some cases, there is a record in both sets that correspond to the same entity, and in other cases an entity is only represented in one set. Formally, let the two sets be **L** and **R**. The cross product of the two sets,

$$\mathbf{L} \times \mathbf{R} = \{(l, r)\,;\, l \in \mathbf{L}, r \in \mathbf{R}\}, \tag{4.1}$$

can be partitioned into two sets: the entities that exist in both sets, the matched records, are

$$\mathbf{M} = \{(l, r)\,;\, l = r, l \in \mathbf{L}, r \in \mathbf{R}\}, \tag{4.2}$$

and the unmatched records are

$$\mathbf{U} = \{(l, r)\,;\, l \neq r, l \in \mathbf{L}, R \in \mathbf{R}\} \tag{4.3}$$

The field values of the records provide the evidence required to match a pair of records, or not, using a linkage rule that assigns a probability for each decision based on the value of some vector function on a pair of records.

Hernández and Stolfo [42] describe a sorted-neighborhood method for merging records from multiple databases. The approach operates under the assumption that keys can be identified for each record such that errors in the field values do not significantly affect the order of the records when sorted by the key. Once a key is identified, the records are merged into a single list and sorted by the key. A sliding window of *w* records is moved over the list and each record that is added to the sliding window is compared with all other items in the window according to some domain specific equational theory to identify matches.

In sensor networks, redundancy elimination is often used to increase the life of the network, which typically consists of cheap disposable components with limited resources that are discarded when their batteries are exhausted. Typically, these approaches rely on knowing the location of each sensor relative to the others as well as their coverage areas.

Carbunar *et al.* [11] describe an approach for identifying nodes who contribute no additional area to the coverage of the sensor network so that these nodes can be shut down and reactivated later after the batteries of other nodes sensing the same area are exhausted to extend the life of the network. The approach relies on knowing the position and sensing range of each node.

Even if the sensors do not have overlapping coverage areas, they may detect the effects of the same phenomenon, *e.g* an increase in temperature due to a fire. This redundancy may be detected by computing the correlation of the measurements collected by neighboring sensors. Vuran *et al.* [90] describe an approach for leveraging this redundancy to reduce the bandwidth required to transmit data from the sensors to the base station which in turn reduces the power used to transmit data, extending the life of the network. In cases where the data collected by a sensor network is not continuously sent to a base station, but is instead collected periodically, Tilak *et al* [89] describe an approach for using correlation among sensor measurements to increase the storage capacity of the sensor network.

In networked systems, event correlation is used for fault localization. Events refer to conditions that arise when a problem is present in the network and, often, a single problem with the system, a fault, produces a number of events as it propagates through the network. The goal of event correlation is to locate faults by identifying sets of events with a common cause. A number of approaches for event correlation have been proposed, including expert systems, machine learning techniques such as decision trees, and fault propagation models such as causality or dependency graphs. Expert systems tend to be less flexible with respect to changes in the network, where graph based models tend to be more robust since they directly encode the relationships among components of the network [83].

Gruschhke [37] describes a dependency graph approach to event correlation. The graph consists of nodes representing the physical components of the network, *e.g.* routers, and the edges represent functional dependencies between the physical components. When events are generated by the network, the event is placed in the graph at the appropriate node and traverses the graph along the edges to model the propagation of the event through the network. The locations of potential faults in the network are the components represented by the nodes in the graph where the paths of propagating events merge.

We have only originating and terminating records that match on the caller and recipient fields and nearly match on the timestamp field. The redundancy elimination techniques in the sensor network literature rely on the ability to correlate time series data, which is not applicable to our problem. Event correlation in networked systems relies on knowledge of the relationship between components in the system to understand how events propagate through the system. In our problem, we do not have this propagation; each transaction over the network is independent of the others. For these reasons, we turn to the record linkage literature for guidance on matching the originating and terminating records in presence of noise in the timestamp field.

## 4.4 Merging the Data Set

The data set used in this chapter is one month of voice calls from cleaned billing and CDR data. The data contains only records of calls made and received by customers that maintain active accounts for a particular year. By removing all non-customers of the service provider, we eliminate most of the cases where we would expect to find incomplete records (one party is using a landline or the network of another provider), particularly for the CDR data. Additionally, only MOC records in the CDR data containing antennas with known locations are included.

We assume that there is only noise in the `start time` component of the key (this is the only field in which perturbations may be detected since it provides an ordering for the records). Our approach iteratively accounts for increasing amounts of clock skew by adjusting the value of the `start time` field during a join operation.

Conceptually, we start by merging the CDR data. Let $\mathbf{L} = (T, C_f, C_t, A_f)$ represent the MOC table and $\mathbf{R} = (T, C_f, C_t, A_t)$ represent the MTC table, where $T$ is the time the call is made, $C_f$ is the caller, and $C_t$ is the recipient, $A_f$ is the originating antenna and $A_t$ is the terminating antenna. We start with an initial matching set $\mathbf{M}$ where the records of $\mathbf{L}$ and $\mathbf{R}$ have identical primary keys, *i.e.* $\mathbf{L}.T = \mathbf{R}.T \wedge \mathbf{L}.C_f = \mathbf{R}.C_f \wedge \mathbf{L}.C_t = \mathbf{R}.C_t$. We ignore the service field because all records are voice calls. Thus $\mathbf{M}$ is the inner join of the two tables on the primary key:

$$\mathbf{M} = \mathbf{L} \underset{\mathbf{L}.T=\mathbf{R}.T \wedge \mathbf{L}.C_f=\mathbf{R}.C_f \wedge \mathbf{L}.C_t=\mathbf{R}.C_t}{\bowtie} \mathbf{R} \tag{4.4}$$

We can decompose $\mathbf{M}$ into two subtables $\mathbf{M_L}$ and $\mathbf{M_R}$ that give the matched records from the original tables $\mathbf{L}$ and $\mathbf{R}$:

$$\mathbf{M_L} = \pi_{T,C_f,C_t,A_f}(\mathbf{M}) \tag{4.5}$$

and

$$\mathbf{M_R} = \pi_{T,C_f,C_t,A_t}(\mathbf{M}) \,. \tag{4.6}$$

Using these tables, we can extract the sets of unmatched records, $\mathbf{U_L}$ and $\mathbf{U_R}$ from the original tables by taking the set differences:

$$\mathbf{U_L} = \mathbf{L} - \mathbf{M_L} \tag{4.7}$$

and

$$\mathbf{U_R} = \mathbf{R} - \mathbf{M_R}. \tag{4.8}$$

Computing the outer join of $\mathbf{U_L}$ and $\mathbf{U_R}$ gives us a new table, $\mathbf{U}$ consisting of only the unmatched records:

$$\mathbf{U} = \mathbf{U_L} \underset{\mathbf{U_L}.T=\mathbf{U_R}.T \wedge \mathbf{U_L}.C_f=\mathbf{U_R}.C_f \wedge \mathbf{U_L}.C_t=\mathbf{U_R}.C_t}{\times} \mathbf{U_R} \tag{4.9}$$

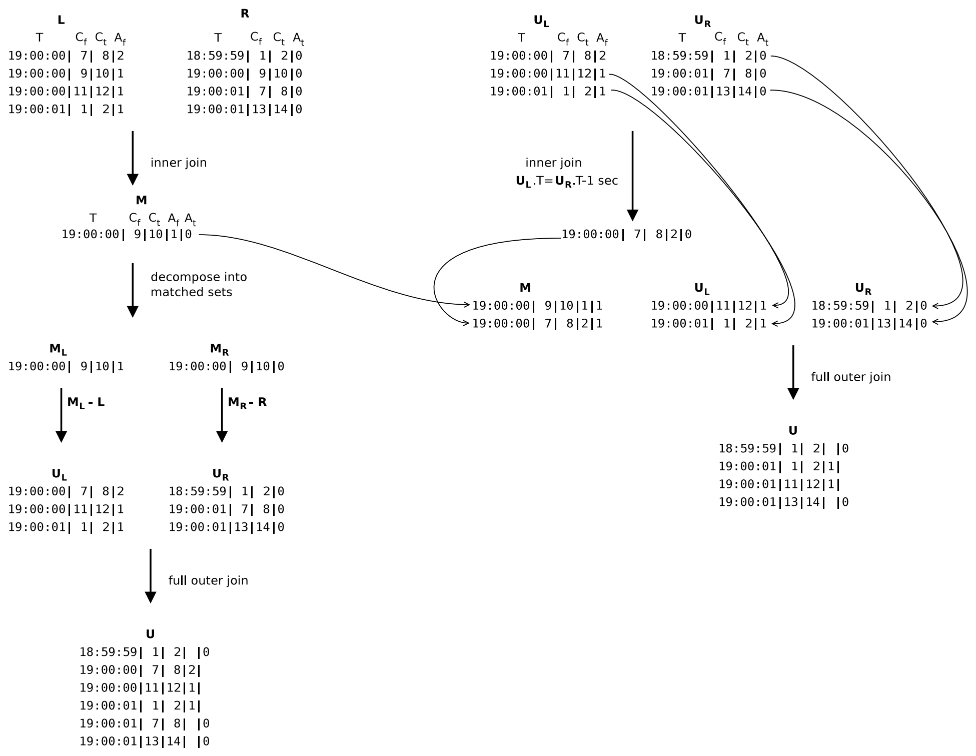
Note that

$$\mathbf{M} \cup \mathbf{U} = \mathbf{L} \underset{\mathbf{L}.T=\mathbf{R}.T \wedge \mathbf{L}.C_f=\mathbf{R}.C_f \wedge \mathbf{L}.C_t=\mathbf{R}.C_t}{\times} \mathbf{R} \tag{4.10}$$

Now that we have identified the matched and unmatched records in **L** and **R**, we attempt to match records in **U** by incrementally searching for records with matching caller and recipient IDs that differ in start time by some $\pm\Delta t \in [1, 120]$. To accomplish this, we join $\mathbf{U_L}$ and $\mathbf{U_R}$ using a condition that adjusts the `start time` of the right table, $\mathbf{U_L}.T = \mathbf{U_R}.T + \Delta t$. So, the join operation is

$$\mathbf{U_L} \underset{\mathbf{U_L}.T=\mathbf{U_R}.T+\Delta t \wedge \mathbf{U_L}.C_f=\mathbf{U_R}.C_f \wedge \mathbf{U_L}.C_t=\mathbf{U_R}.C_t}{\bowtie} \mathbf{U_R} \tag{4.11}$$

Each record in this join is moved from $\mathbf{U_L}$ and $\mathbf{U_R}$ to $\mathbf{M_L}$ and $\mathbf{M_R}$, respectively. By repeating this process over the sequence $\Delta t = +1, -1, +2, -2, \ldots,$ we ensure that we match records in **L** to a record in **R** with the minimum possible $|\Delta t|$, and by updating the sets $\mathbf{U_L}$ and $\mathbf{U_R}$, we ensure that each record is only matched up to one time. Figure 4.1 provides an illustrative example of this process.


inner join
decompose into matched sets
$\mathbf{M_L}$ - $\mathbf{L}$
$\mathbf{M_R}$ - $\mathbf{R}$
full outer join
inner join
$\mathbf{U_L}$.T=$\mathbf{U_R}$.T-1 sec
full outer join

**L**
```
   T      C_f C_t A_f
19:00:00| 7| 8|2
19:00:00| 9|10|1
19:00:00|11|12|1
19:00:01| 1| 2|1
```

**R**
```
   T      C_f C_t A_t
18:59:59| 1| 2|0
19:00:00| 9|10|0
19:00:01| 7| 8|0
19:00:01|13|14|0
```

**M**
```
   T      C_f C_t A_f A_t
19:00:00| 9|10|1|0
```

$\mathbf{M_L}$
```
19:00:00| 9|10|1
```

$\mathbf{M_R}$
```
19:00:00| 9|10|0
```

$\mathbf{U_L}$
```
19:00:00| 7| 8|2
19:00:00|11|12|1
19:00:01| 1| 2|1
```

$\mathbf{U_R}$
```
18:59:59| 1| 2|0
19:00:01| 7| 8|0
19:00:01|13|14|0
```

**U**
```
18:59:59| 1| 2| |0
19:00:00| 7| 8|2|
19:00:00|11|12|1|
19:00:01| 1| 2|1|
19:00:01| 7| 8| |0
19:00:01|13|14| |0
```

$\mathbf{U_L}$
```
   T      C_f C_t A_f
19:00:00| 7| 8|2
19:00:00|11|12|1
19:00:01| 1| 2|1
```

$\mathbf{U_R}$
```
   T      C_f C_t A_t
18:59:59| 1| 2|0
19:00:01| 7| 8|0
19:00:01|13|14|0
```

```
19:00:00| 7| 8|2|0
```

**M**
```
19:00:00| 9|10|1|1
19:00:00| 7| 8|2|1
```

$\mathbf{U_L}$
```
19:00:00|11|12|1
19:00:01| 1| 2|1
```

$\mathbf{U_R}$
```
18:59:59| 1| 2|0
19:00:01|13|14|0
```

**U**
```
18:59:59| 1| 2| |0
19:00:01| 1| 2|1|
19:00:01|11|12|1|
19:00:01|13|14| |0
```

Figure 4.1. An illustration of the the merging process. The left column shows the process of decomposing the MOC table, **L**, and the MTC table, **R** into the matched, **M**, and unmatched, **U**, sets. The right column shows one step of the process where we account for clock skew in the MTC table and migrate the newly match records to the matched set **M**.

In practice, we build a working table **W**. This table is the full outer join of **L** and **R**,

$$\mathbf{W} = \mathbf{L} \underset{\mathbf{L}.T=\mathbf{R}.T \wedge \mathbf{L}.C_f=\mathbf{R}.C_f \wedge \mathbf{L}.C_t=\mathbf{R}.C_t}{⟗} \mathbf{R}, \tag{4.12}$$

with an additional field, $S$, that indicates the skew of the record. So $\mathbf{W} = (T, C_f, C_t, A_f, A_t, S)$. Initially, the skew field is 0 for complete records, *i.e.* the records in the inner join, and null for the remaining records:

$$S \leftarrow \begin{cases} 0 & \sigma_{A_f \neq \omega \wedge A_t \neq \omega}(\mathbf{W}) \\ \omega & \sigma_{A_f = \omega \vee A_t = \omega}(\mathbf{W}). \end{cases} \tag{4.13}$$

Records with a null skew field are in the unmatched set, **U**; all other records are in **M**. For each $\Delta t = 1, -1, 2, -2, \ldots 120, -120$, we compute the inner join of $\mathbf{U_L}$ and $\mathbf{U_R}$:

$$\mathbf{M}' = \rho_{\sigma_{S=\omega \wedge A_f \neq \omega}(\mathbf{W})}(\mathbf{U_L}) \underset{\mathbf{U}_L.T=\mathbf{U}_R.T+\Delta t \wedge \mathbf{U}_L.C_f=\mathbf{U}_R.C_f \wedge \mathbf{U}_L.C_t=\mathbf{U}_R.C_t}{\bowtie} \rho_{\sigma_{S=\omega \wedge A_t \neq \omega}(\mathbf{W})}(\mathbf{U_R}) \tag{4.14}$$

For each element in $\mathbf{M}'$, we update the skew: $S \leftarrow \Delta t$.

For each subsequent step in the merge, we update the skew field for both of the matched records. Once we have updated the skew field for the largest $\Delta t$, we add an additional field that gives the corrected time. The corrected time field is set for all records in the table according to the following rules:

- If the skew field is 0, null, or the record is from the left table the corrected time is equal to the start time.

- The remaining records, *i.e.* the records from the right table with a non-null, nonzero skew, the corrected time field is the start time field plus the interval given by the skew field.

Formally, for each record in **W** after the correction,

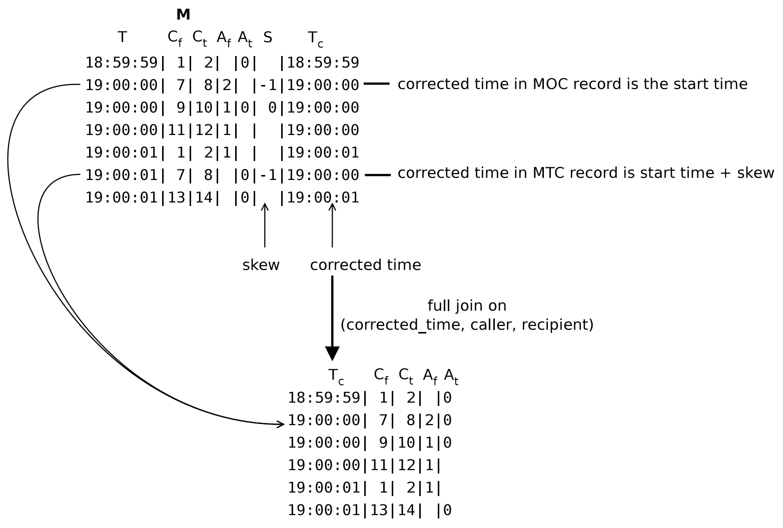
$$T_c \leftarrow \begin{cases} T & \sigma_{S=0 \vee S=\omega \vee A_f \neq \omega}(\mathbf{W}) \\ T + \Delta t & \sigma_{A_t \neq \omega \wedge S \neq 0 \wedge S \neq \omega}(\mathbf{W}). \end{cases} \tag{4.15}$$

The resulting table allows us to extract both the corrected and uncorrected data. The uncorrected data is straightforward to extract as all modifications have been made to the newly added field, leaving all of the original field values untouched; therefore, a simple `SELECT` statement obtains the data. The corrected data can be obtained by performing the full outer join using the corrected time, caller, and recipient. We can determine the left and right tables for the join by looking for the presence of non-key field values. Consider the merged CDR data: the MOC records must have a non-null value in $A_f$ and the MTC records must have a non-null value in $A_t$, so we obtain the corrected table with the following query:

$$\rho_{\pi_{T_c,C_f,C_t,A_f}\left(\sigma_{A_f \neq \omega}(\mathbf{W})\right)}(\mathbf{L}) \underset{\mathbf{L}.T_c=\mathbf{R}.T_c \wedge \mathbf{L}.C_f=\mathbf{R}.C_f \wedge \mathbf{L}.C_t=\mathbf{R}.C_t}{\times} \rho_{\pi_{T_c,C_f,C_t,A_t}\left(\sigma_{A_t \neq \omega}(\mathbf{W})\right)}(\mathbf{R}) \tag{4.16}$$

## 4.5 Evaluation Method

We use the Kullback-Leibler divergence to measure the distance between the inter-call time distribution of the merged CDR data and the call data. The Kullback-Leibler divergence, or relative entropy, is a result from information theory [51]. Suppose we have an alphabet $\mathbf{X} = \{x_1, x_2, \ldots, x_n\}$ where each element, $x_i$ occurs with probability $p_i$. The information provided by each element $x_i$, in bits, is $\lg(p_i)$. Intuitively, elements that are more likely contain less information (and can be represented with fewer bits) than elements that appears rarely. The entropy of $X$, which can be viewed as a random


M
T $C_f$ $C_t$ $A_f$ $A_t$ S $T_c$
corrected time in MOC record is the start time
corrected time in MTC record is start time + skew
skew
corrected time
full join on
(corrected_time, caller, recipient)
$T_c$ $C_f$ $C_t$ $A_f$ $A_t$

```
18:59:59| 1| 2| |0|   |18:59:59
19:00:00| 7| 8|2| |-1|19:00:00
19:00:00| 9|10|1|0| 0|19:00:00
19:00:00|11|12|1| |   |19:00:00
19:00:01| 1| 2|1| |   |19:00:01
19:00:01| 7| 8| |0|-1|19:00:00
19:00:01|13|14| |0|   |19:00:01
```

```
18:59:59| 1| 2| |0
19:00:00| 7| 8|2|0
19:00:00| 9|10|1|0
19:00:00|11|12|1|
19:00:01| 1| 2|1|
19:00:01|13|14| |0
```

Figure 4.2. An illustration of the process of extracting corrected data from a table with supplementary `skew` and `corrected time` fields.

variable, is the expected information over the entire alphabet:

$$H = \sum_{x_i \in X} -p_i \log p_i \tag{4.17}$$

[19, 60, 81].

Suppose we have the same alphabet $X$ as before; however, we assume each element $x_i$ now occurs with a probability $q_i$ while the true probability is still $p_i$. For each $x_i$, the difference in information is $log\frac{p_i}{q_i}$, and the relative entropy, or Kullback-Leibler divergence is the average difference in information over the set $X$:

$$D_{\mathrm{KL}} = \sum_{x_i \in X} p_i \log \frac{p_i}{q_i} \tag{4.18}$$

Intuitively, this gives the average number of additional bits required to encode a symbol using the assumed probabilities $q_i$ rather than the actual probabilities $p_i$ [19, 51].

We considered using the Kolmogorov-Smirnov test (KS-test); however, this proved problematic. Suppose we have two cumulative distribution functions $P$ and $Q$. The KS-test statistic, $d$, is the maximum distance between these two distributions:

$$d = \max_{x \in X} |P(x) - Q(x)| \tag{4.19}$$

The critical values of $d$ for a given confidence interval depend on the sample size (the larger the data size, the smaller the critical value) [59]. When we examined results generated using the KS-test statistic, we found that the best stopping point was, in reality, too small. When looking at plots of the distribution closest to the distribution for the call data, that there was obvious noise in the tail. We concluded that the KS-test is not expressive enough, since it only considers the maximum distance between the cumulative distributions and a method using a pairwise comparisons of the points in the distribution would be more appropriate. For this reason we use the Kullback-Leibler divergence.

To determine the need for clock skew correction, we examine the intercall probability distribution over a two minute interval. Consider the directed call multi-graph where the vertices represent phones and the calls are directed edges from the caller to the recipient and each edge is labeled with the time the call was made. For the set of directed edges from a particular vertex to another particular vertex, we find the intervening time between each pair of consecutive calls. Figure 4.3 illustrates how we compute this distribution.

We use the intercall time distribution to evaluate our approach because it is a measure that is sensitive to potential redundancy introduced by noise in the timestamps. In the cases where noise prevents two records from being merged, the resulting data set will have the two records with a small time interval between them, skewing the intercall

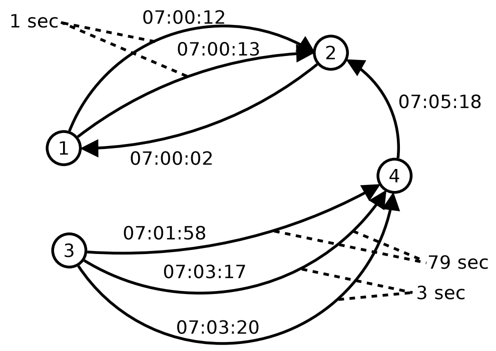


Figure 4.3. Example showing how the intercall time distribution is generated. There are three pairs of consecutive calls in this graph: 1 pair of edges from 1 to 2, separated by 1 second and 2 pairs of edges from 3 to 4, (07:01:58, 07:03:17) and (07:03:17, 07:03:20), separated by 79 seconds and 3 seconds, respectively. The other two edges are unique in their source and destination, so there is no consecutive call in either case; therefore, neither of these edges contributes to the intercall time distribution. Dividing by the number of data points gives the empirical probability density function (each of 1, 3, and 79 seconds occurs with probability $\frac{1}{3}$).

time distribution to the left. In theory, the call and CDR data both describe the same set of events. We therefore expect that there is significant overlap and the properties of the data are similar. Our experience with the data has shown that the call data is significantly cleaner than the CDR data, and because we do not have to perform a merge to obtain it, as we would with the CDR data, we use the intercall time distribution from this data set as our baseline.

## 4.6 Results

In this section, we describe the behavior of the KL-divergence as we increase $|\Delta t|$ when performing the two merges and its use for determining the stopping point of the merge process. We also examine the effect of the merging process on the fraction of matched and unmatched records in the data sets.

The graph at the top of figure 4.4 shows the KL-divergence of the merged CDR intercall time distribution from the call data intercall time distribution for $|\Delta t| \in [1, 120]$, and the inset shows the same value over the range $|\Delta t| \in [60, 120]$ to expose detail near the minimum. We select $|\Delta t| = 81$ seconds, the minimum, as our stopping point and continue our analysis with the data correction applied up to this point.

The center row of figure 4.4 shows the intercall time distribution of the merged data set corrected up to $|\Delta t| = 81$ in the center with the distribution for the uncorrected merged data on the left and the baseline call data distribution on the right. The similarity between the corrected distribution and the baseline is striking.

The bottom row of figure 4.4 show the intercall time distribution for the merged CDR data at various points in the correction process: two points before the stopping point (up to $|\Delta t| = 20, 45$) and one point after (up to $|\Delta t| = 120$). In the former cases, there is visible noise in the distributions for $|\Delta t| < 81$; however, the distribution of the latter case looks quite similar to the base line. In fact, its KL-divergence from the baseline is only slightly larger at $|\Delta t| = 120$ than the stopping point ($7.45 \times 10^{-3}$ and $7.23 \times 10^{-3}$, respectively).

The graph at the top of figure 4.5 shows the KL-divergence of the merged call and CDR intercall time distribution from that of the call time alone for $|\Delta t| \in [1, 120]$, and the inset shows the same value over the range $|\Delta t| \in [45, 120]$ to expose detail near the minimum. There is consistent decrease in the range $|\Delta t| = [1, 55]$, however, in $[55, 81]$, the behavior of the function is sporadic. This sporadic behavior, however, occurs over a small range of values of the KL-divergence ($[1.5340 \times 10^{-2}, 1.5385 \times 10^{-2}]$, a difference of $4.5 \times 10^{-5}$).

The center row of figure 4.5 shows the intercall distribution of the merged call and CDR data corrected up to $|\Delta t| = 62$ in the center with the uncorrected merged data on

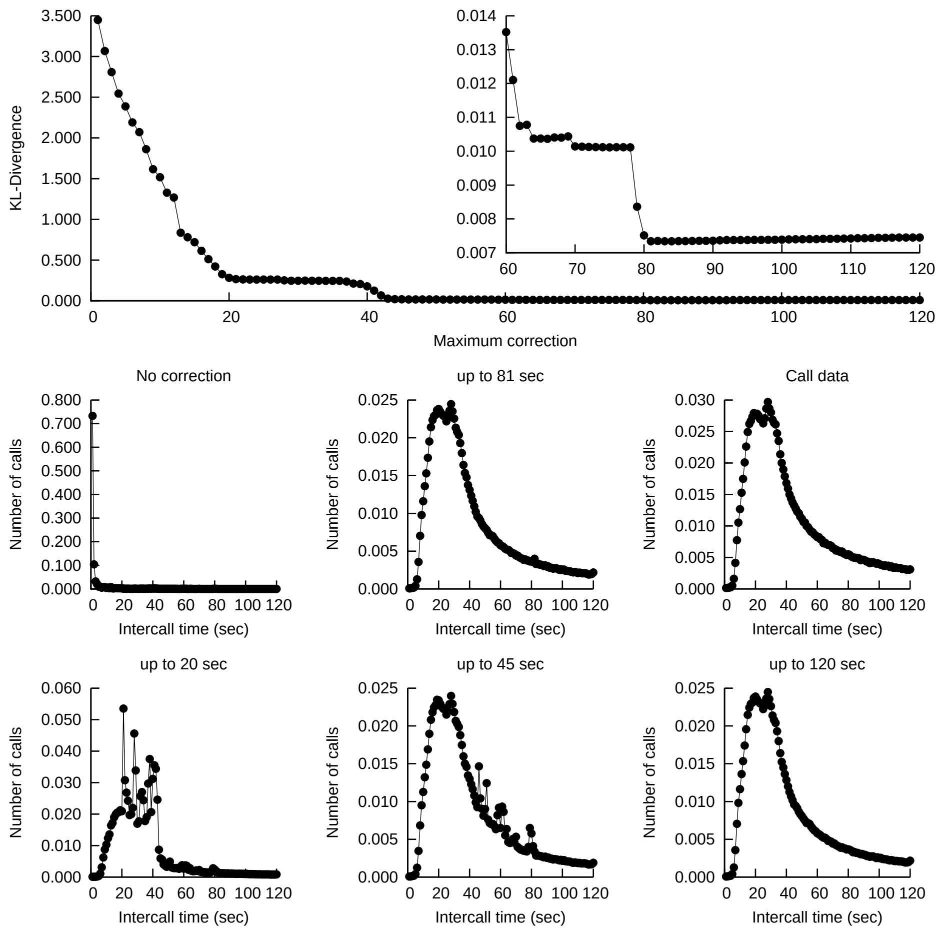


Figure 4.4. The top figure shows Kullback-Leibler divergence between the merged CDR intercall time and call data intercall time distributions as the correction procedure progresses. The middle right figure shows the KL-divergence for the larger half of the range to show detail around the minimum at 81 seconds. The middle left and center show the uncorrected merged CDR intercall time distribution and the call data intercall time distribution that is used for the baseline, respectively. The bottom row shows snapshots of corrected merged CDR intercall time as the correction process proceeds, at 20, 45, 81, and 120 seconds.

the left and the baseline call distribution on the right. Again, the similarity between the corrected distribution and the baseline is compelling.

The bottom row of figure 4.5 shows the intercall time distribution for the merged call and CDR data at various points in the correction process: two points before the stopping point (up to $|\Delta t| = 20, 40$) and one point after (up to $|\Delta t| = 90$). Again, when we stop the process before reaching the minimum KL-divergence values, there is visible noise in the distribution after the stopping point and the distribution shown that is beyond the stopping point appears very similar to the baseline distribution.

We are also interested in how many records are merged in this process, so we measure the fraction of records in the sets $\mathbf{M}$, $\mathbf{U_L}$, and $\mathbf{U_R}$ before and after each merge. When merging the CDR data, $\mathbf{U_L}$ consists of records only in the MOC table, $\mathbf{U_R}$ consists of records only in the MTC table, and $\mathbf{M}$ consists of records that appear in both tables. Figure 4.6 shows the fraction of records belonging to each set before and after the correction (of up to $|\Delta t| = 81$). Applying the correction reduces the number of records in the CDR table 15%, from 115,778,284 records to 98,128,532, increases the number of records in $\mathbf{M}$ 35%, from 50,477,486 to 68,127,208, and reduces the number of records in $\mathbf{U_L}$ and $\mathbf{U_R}$ 62%, from 28,518,775 to 10,869,033, and 48%, from 36,782,013 to 19,132,291, respectively.

Figure 4.7 shows the fraction of records in each set $\mathbf{M}$, $\mathbf{U_L}$, and $\mathbf{U_R}$ when merging the call and CDR data before and after the correction is applied. Applying the correction reduces the number of records in the table 3%, from 102,701,857 records to 99,878,208, increases the number of records in $\mathbf{M}$ 5%, from 54,173,052 to 56,996,701, and reduces the number of records in $\mathbf{U_L}$ and $\mathbf{U_R}$ 62%, from 4,872,325 to 1,749,676, and 4%, from 43,955,480 to 41,131,831, respectively. We see right away that the clock skew isn't as significant in this case because $\mathbf{U_L}$ is very small (only about 4% of the records), and we

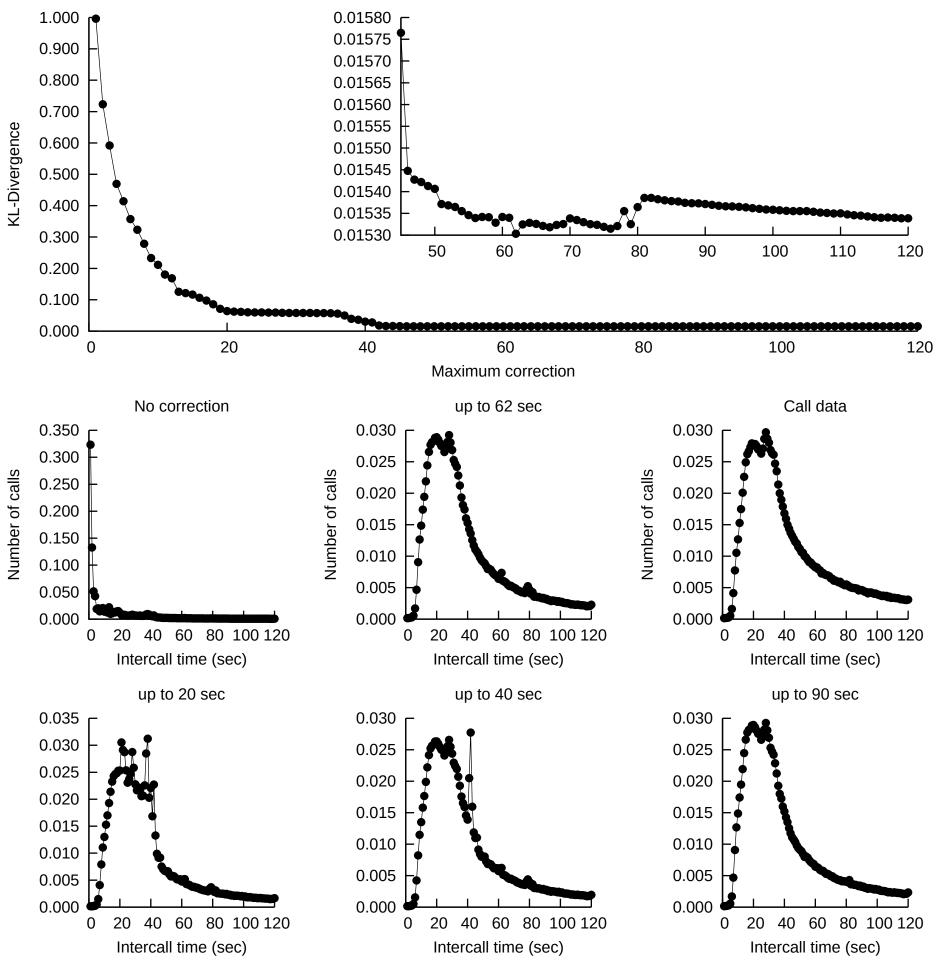


Figure 4.5. The top figure shows Kullback-Leibler divergence between the merged usage intercall time and call data intercall time distributions as the correction procedure progresses. The middle right figure shows the KL-divergence for the larger half of the range to show detail around the minimum at 81 seconds. The middle left and center show the uncorrected merged usage intercall time distribution and the call data intercall time distribution that is used for the baseline, respectively. The bottom row shows snapshots of corrected merged CDR intercall time as the correction process proceeds, at 20, 40, 62, and 120 seconds.

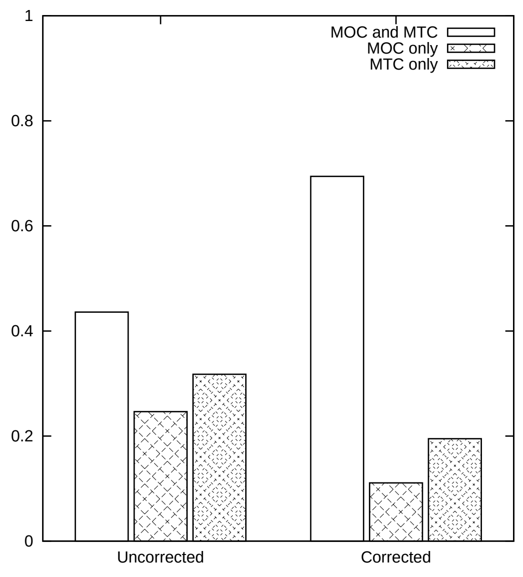


Figure 4.6. The fraction of records in the matched and unmatched sets before and after the merge correction of the MOC and MTC tables. "MOC and MTC" records correspond to $\mathbf{M}$, "MOC only" corresponds to $\mathbf{U_L}$, and "MTC only" corresponds to $\mathbf{U_R}$.

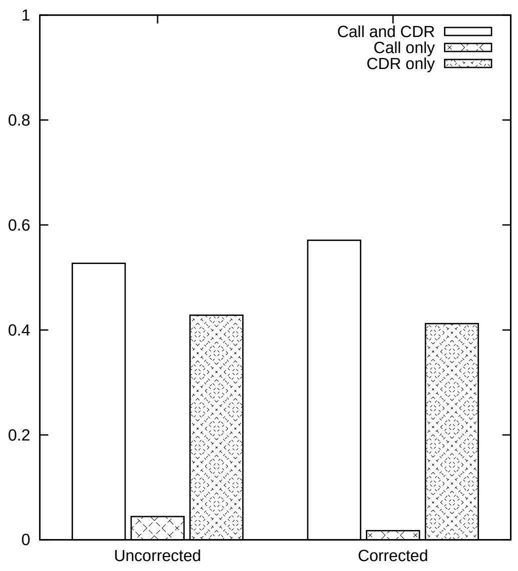


Figure 4.7. The fraction of records in the matched and unmatched sets before and after the correction of the billing and CDR tables. “Billing and CDR” corresponds to $\mathbf{M}$, “Billing only” corresponds to $\mathbf{U_L}$, and “CDR only” corresponds to $\mathbf{U_R}$.

are able to merge more than half of these records by applying the correction.

## 4.7 Discussion and Conclusion

In this chapter, we have shown that there is some noise in the timestamps associated with the usage records that introduces redundancy if the components of the data are merged naively, and we have described an approach for removing this redundancy.

Whether this noise has an impact on the results of work using these data depends on the particulars of the study. Many of the major studies using these data are not affected by this problem. In these cases, no merge is performed. Onnela *et al* [63] use only the billing data to build a weighted network. González *et al* [33] and Wang *et al* [91] use only the originating (MOC) records of the CDR data.

In cases where the tables are merged, unweighted graphs built using the data will not be affected since the redundancy does not affect whether a particular edge is present or not. Similarly, graph with weighted edges based on some attribute only present in one table, *e.g.* cost or duration, will not be affected because the redundant records do not provide information to be added to the weights. Problems with this redundancy do, however, arise when weighting the edges of a graph by the number of times that edge appears in the data or when counting the overall number of calls.

# CHAPTER 5

# BACKGROUND ON DATA CLUSTERING

The goal of data clustering is to identify meaningful groups in data in which similar data items belong to the same group and dissimilar items belong to different groups. Traditionally, groups were identified subjectively, by eye; however, this is only feasible for data with only 2 or 3 dimensions, due to the limits of human perception, and tends to yield inconsistent results. As datasets have grown in the number of features, many clustering algorithms have been implemented in a number of different areas of research. Data clustering is an approach for exploratory data analysis that is widely applicable to a number of fields [46, 49][1].

The clustering problem is defined as follows: let a data set **D** consist of a set of data items $(\vec{d}_1, \vec{d}_2, \ldots, \vec{d}_n)$ such that each data item is a vector of measurements, $\vec{d}_i = (d_{i,1}, d_{i,2}, \ldots, d_{i,m})$. Using some measure of dissimilarity, $D(\vec{d}_i, \vec{d}_j)$, group the data into sets $C_1, C_2, \ldots, C_k$ based on some criteria [35].

At the highest level, clustering algorithms can be divided into two types: partitional and hierarchical. Partitional algorithms divide the data into some number of disjoint sets. Hierarchical algorithms divide the data into a nested set of partitions. They may either take a top-down approach, in which the hierarchy is generated by splitting clusters, or a bottom-up approach, in which the hierarchy is generated by merging clusters

[1]Portions of this chapter are to appear in *Intelligent Techniques for Warehousing and Mining Sensor Network Data*, ed. A. Cuzzocrea, 2009 [68]

[47]. Incremental and stream clustering algorithms have also been developed. In this section, we present a brief overview of clustering algorithms.

## 5.1 Partitional Clustering

Partitional clustering divides the data set into some number, often a predefined number, of disjoint subsets. Some of the most widely know partitional clustering algorithms are iterative relocation algorithms. Such algorithms are constructed by defining 1) a method for selecting the initial partitions, and 2) a method for updating the partitions [35]. *K*-means and expectation maximization use this approach. In *k*-means, *k* data points are randomly selected as centroids, each data point is assigned to the nearest centroid, and the centroids are recomputed. This process is repeated until no data items move to a different cluster. The expectation maximization algorithm is similar, the initial clusters are random Gaussians, each data item is assigned to the Gaussian with the highest probability of generating the point, and the Gaussians are iteratively recomputed [94].

A partitional clustering can also be computed using a minimum spanning tree approach. Let the data set be represented as a complete graph where the data items are represented as vertices and the edges weights are the distance between the two connected data items. From this graph, compute the minimum spanning tree. To obtain $k$ clusters, remove the $k-1$ edges with the largest weights [35].

## 5.2 Hierarchical Clustering

Hierarchical clustering algorithms generate a nested set of partitions. There are two common types of hierarchical algorithms: agglomerative and divisive [49].

Most hierarchical algorithms are agglomerative, meaning that they take a bottom-up approach in which the data set is initially partition in to $n$ clusters, each of which contain one data item. The algorithm then iteratively merges the two nearest clusters until all of the data belongs to a single cluster. The various agglomerative algorithms vary in the definition of the difference between two clusters [49]. Well known agglomerative algorithms include the single and complete link algorithms. For these algorithms, the distance between clusters is the minimum (equation 5.1) and maximum distance (equation 5.2) between data items of the two clusters, respectively [47].

$$D(C_i, C_j) = \min_{\vec{d_i} \in C_i, \vec{d_j} \in C_j} (d(\vec{d_i}, \vec{d_j})) \tag{5.1}$$

$$D(C_i, C_j) = \max_{\vec{d_i} \in C_i, \vec{d_j} \in C_j} (d(\vec{d_i}, \vec{d_j})) \tag{5.2}$$

The agglomerative nesting algorithm described by Kaufman and Rousseeuw [49] uses a distance measure weighted by the inverse of the product of the sizes of the clusters:

$$D(C_i, C_j) = \frac{1}{|C_i||C_j|} \sum_{\vec{d_i} \in C_i, \vec{d_j} \in C_j} d(\vec{d_i}, \vec{d_j}) \tag{5.3}$$

Divisive algorithms proceed from the top-down, starting with all data items in a single cluster and repeatedly split clusters until each cluster has only a single member. These algorithms are less common and, when implemented naively, are much more computationally expensive. Consider the first step of an agglomerative clustering algorithm: the distance between each pair of data items must be computed. This requires $O(n^2)$ calls to the distance function. Now consider the task of examining all possible ways of splitting a cluster into two: there are $O(2^n)$ possible partitions to consider. The divisive analysis (DIANA) algorithm uses the data item in a cluster that is most dissimilar from the other members of the cluster to seed a new cluster and eliminate the need for exhaustive evaluation of the possible partitions. At each step, the largest cluster, the cluster with the largest distance between two of its members, is split. The data item in this cluster with the largest average dissimilarity from the other items in the cluster is used to seed the new cluster, and each data point for which the average dissimilarity from the items in the new cluster is less than the average dissimilarity from the items remaining in the original cluster is moved to the new cluster. The process is repeated until no data points remain in the original cluster with a smaller average dissimilarity from the data in the new cluster [49]. Another divisive approach uses some criteria to select a cluster to split, *e.g.* the cluster with the highest variance, and applies a partitional algorithm, such as *k*-means to split the cluster in two [76].

## 5.3 Hybrid Algorithms

Hybrid clustering combines two clustering algorithms. Cheu *et al* [15] examine the use of iterative, partitional algorithms such as *k*-means, which tend to be fast, as a method of reducing a data set for hierarchical, agglomerative algorithms, such as complete-link, that tend to have high computational complexity. Chipman and Tibshiran [17] combine agglomerative algorithms, which tend to do well at discovering small clusters, with top-down methods, which tend to do well at discovering large clusters. Surdeanu *et al* [85] propose a hybrid clustering algorithm for document classification that uses hierarchical clustering as a method for determining initial parameters for expectation maximization.

## 5.4 Incremental Clustering

Incremental algorithms consider each example once, immediately deciding either to place it in an existing cluster or to create a new cluster. These algorithms tend to be fast, but are also often order dependent [47]. The leader algorithm is a simple incremental clustering algorithm in which each cluster is defined by a single data item—the first item assigned to the cluster. For each data example, if the example is within a user specified distance of the defining item of the closest cluster, the example is assigned to that cluster; otherwise, the example becomes the defining example of a new cluster [41].

Charikar *et al.* [12] describes incremental clustering algorithms that maintain a fixed number of clusters with certain properties as new data points arrive. Two algorithms are described; each uses a $t$-threshold graph in which each pair of points in the data set is connected with an edge if and only if the distance between the two points is less than some threshold $t$. The algorithms are a two step processes in which: 1) the existing clusters are merged as necessary to ensure that there are no more than $k$ clusters and 2) new data items are processed until there are more that $k$ clusters. The updates are constrained by invariants, and in cases where a point cannot be added to any existing cluster without violating an invariant, it is added to a new cluster. The doubling algorithm constrains the maximum radius (the minimum sphere containing all points in the cluster) and the minimum inter-cluster distance. When the number of clusters exceeds $k$, arbitrarily selected clusters are merged with their neighbors until the number of clusters is less than $k$. The clique partition algorithm enforces a maximum radius and diameter (maximum distance between two points in a cluster) during the update phase and merges are performed by computing a clique partition of the threshold graph and combining the clusters in each partition.

## 5.5 Clustering Algorithms for Streaming Data

Some methods have been developed for clustering data streams. Guha *et al* [38] present a method based on *k*-mediods—an algorithm similar to *k*-means. The clusters are computed periodically as the stream arrives, using a combination of the streaming data and cluster centers from previous iterations to keep memory usage low. Aggarwal *et al* [3] present a method that takes into account the evolution of streaming data, giving more importance to more recent data items rather than letting the clustering results be dominated by a significant amount of outdated data. The algorithm maintains a fixed number of *micro-clusters*, which are statistical summaries of the data throughout the stream. These micro-clusters serve as the data points for a modified *k*-means clustering algorithm.

## 5.6 Cluster Evaluation

Since clustering is an unsupervised learning method, it is often not feasible to use a training and test set for evaluation. There are three major approaches for evaluating the results of clustering algorithms: external, internal, and relative methods [39].

External measures require that a labelled training set exists to compare the results of the algorithm with the grouping determined by the training set labels. Internal measures use aspects of the data to evaluate the results. These aspects include compactness, which can be evaluated using sum of squared error or average pairwise intra-cluster distance, connectedness, which can be measured by comparing the clustering results with the result of *k* nearest neighbors, and separation, which can be measured using the average weighted inter-cluster distance [40]. Relative measures compare the results of a set clustering schemes [39].

## 5.7 Summary

In the next two chapters, we use the concepts described in this chapter to build tools for detecting anomalies in multivariate streaming data. In chapter 6, we use a hybrid algorithm consisting of a hierarchical algorithm, complete link, and an incremental algorithm, leader, to detect events. In chapter 7, we use a hierarchical feature clustering algorithm over a sliding window to detect events and their locations.

# CHAPTER 6

# ONLINE CLUSTER ANALYSIS OF SPATIALLY PARTITIONED PHONE USAGE DATA

## 6.1 Introduction

In this chapter[1], we describe an online clustering algorithm that combines complete link clustering and a variant of the leader algorithm. Our algorithm differs from a number of other online clustering algorithms in that we do not fix the number of clusters throughout; rather, we use heuristics, similar to those in existing algorithms, to allow the cluster set to grow and contract as more data arrives.

It was our intention to use this algorithm as an approach for detecting events in phone data for the Wireless Phone Based Emergency Response System; however, at the time of the initial development we did not have a suitable data set containing known emergency events. Now that we are in possession of such data sets, we realize that this type of clustering is problematic for this application and that a feature clustering approach, which we present in the next chapter, is better suited for our goals. Despite this, the development of this algorithm has yielded some interesting insights, and we present the work here to share these insights.

[1]An earlier version of this work was presented and received the best student paper awards at the 2006 Conference of the North American Association for Computational Social and Organization Sciences [64] and was subsequently published in the journal *Computational & Mathematical Organization Theory* [65].

## 6.2 Related Work

Traditional clustering algorithms require random access over the full data set; however, this may not be feasible due to the nature of the data. In cases where the data set is too large for main memory or arrives as a fast data stream, more advanced, incremental algorithms are required. Typically these algorithms take a hybrid approach. The data items are initially clustering into a large number of small clusters and these clusters are used as a reduction of the data set to improve the performance of the main clustering algorithm, which may be one of the traditional algorithms adapted to group clusters rather than points.

Zhang *et al.* [95] describe the Balanced Iterative Reducing and Clustering using Hierarchies (BIRCH) algorithm. In this work, they define the clustering feature which is later used in a number of other algorithms. Suppose we have a cluster **C** containing data points $\{\vec{d}_1, \vec{d}_2, \ldots, \vec{d}_n\}$. The clustering feature for this cluster is $(n, \vec{s}_1, \vec{s}_2)$ where $\vec{s}_1$ is vector of feature sums for the data points, $s_{1,i} = \sum_{j=1}^{n} d_{i,j}$ and $\vec{s}_2$ is the vector of squared sums of the features, $s_{2,i} = \sum_{j=1}^{n} d_{i,j}^2$. BIRCH uses a tree structure to efficiently build the clustering features that reduce the data set and finalizes the clustering by applying an agglomerative clustering algorithm to the clustering features.

Bradley *et al.* [8] describe an approach for scaling $k$-means to very large data sets by using two types of data compression. The algorithm employs a buffer that, at each step, reads and processes a sample from the data set, updates the cluster model, and then applies the compression techniques to eliminate the need to store points that are unlikely to move to a different cluster and sets of points that are likely to move from one cluster to another as a group. In the latter case, the points are collapsed into a sub-cluster described by a clustering feature. Farnstrom *et al* [26] simplified this algorithm to repeatedly apply the clustering algorithm to the current points in the buffer and the existing set of clusters, clearing the data buffer at each step.

Aggarwal *et al.* [3] describes a hybrid approach for stream clustering that uses an extension of clustering features which are updated with the arrival of each point to provide a manageable sized data set for the application of a second clustering algorithm. The algorithm for updating the “microclusters” is similar the leader algorithm; however, in this case, a fixed number of clusters in maintained and the threshold for accepting a data point into an existing cluster is based on the root mean squared deviation of the data items in the cluster. The microclusters are stored using an extension of feature clusters. Two time components are added, $t_1$ and $t_2$, which give the sum and squared sum of the timestamps for the data items belonging to the microcluster, respectively. When a new data item does not fit into any cluster, based on the root mean squared deviation, it is placed in a new cluster, and either an existing cluster is deleted or two clusters are merged. Using the two time values in the extended feature clusters, the cluster with the least “relevance” is identified where relevance is a measure of whether the cluster is still acquiring data items. If the minimum relevance is below some threshold, the cluster is removed, otherwise, the two nearest clusters are merged.

## 6.3 Hybrid Clustering Algorithm

We use a hybrid clustering algorithm designed to expand and contract the number of clusters in the model based on the data rather than assigning an fixed number of clusters specified *a priori*. The basic idea behind the algorithm is to use complete link to establish a set of clusters and then use the leader algorithm in conjunction with statistical process control to update the clusters as new data arrives.

Statistical process control [7] aims to distinguish between “assignable” and “random” variation. Assignable variations are assumed to have low probability and indicate some anomaly in the underlying process. Random variations, in contrast, are assumed to be quite common and to have little effect on the measurable qualities of the process. These two types of variation may be distinguished based on the difference in some measure on the process output from the mean, $\mu$, of that measure. The threshold is typically some multiple, $l$, of the standard deviation, $\sigma$. Therefore, if the measured output falls in the range $\mu \pm l\sigma$, the variance is considered random; otherwise, it is assignable.

We use the clustering feature described by Zhang *et al* [95] to maintain the sufficient statistics of the clusters: for each cluster, we store the sum and sum of squares of field values for each example in the cluster and the number of elements in the cluster. We use the same approach for identifying the cluster membership of a new example as in [3]; however, we handle cluster retirement differently.

For each new data item, we locate the nearest cluster. If the new data item is within some multiple of the root mean squared deviation of the nearest cluster, the item is added to that cluster; otherwise a new cluster is created. We call this threshold the *membership threshold*, $t_m$. In cases, where the nearest cluster has only 1 data item we add the item if it is closer to the nearest cluster than the nearest cluster is to its nearest neighbor. If a new cluster is created, we search for overlapping clusters and merge them. We consider two clusters to be overlapping if the distance between the centroid of each cluster is within some multiple of the root mean squared deviation of the other cluster. We call this the *overlap threshold*, $t_o$.

## 6.4 Experimental Setup

We apply the above clustering algorithm to two data sets derived from call record data of a cellular service provider. The raw data consists of the time at which the call is made and the tower through which the call originates. Using GIS data associated with the towers, we partition the data into a set of features that give the number of calls per 10 minute interval over each postal code for a particular city. We use 2 distinct one month data sets.

We compute the initial clusters for each month using the first day of data, varying the number of sets in this partition and then proceed with the online portion of the algorithm. We evaluate the clusterings using sum squared error to measure the compactness of the clusters and average distance between centroids to measure the separation. Since the hybrid algorithm does not keep track of the cluster membership of each data item, we assign each data item to the cluster with the nearest center at the end of the run. We compare these results with those of the offline complete link using the same approach of assigning data items to clusters described by the centroids computed using the data items and their cluster assignments.

## 6.5 Results

Table 6.1 shows the results of our algorithm on the first dataset. For each trial, the first day of data is partitioned into the initial set of clusters, $C_i \in [2, 20]$ in number using membership and overlap thresholds of 6 times the root mean squared deviation (RMSD). For this data set, the algorithm tends to find a small number of clusters, 3-4. As $C_i$ increases, the results of the clustering algorithm stabilizes such that the resulting set of clusters is identical for $C_i \in [9, 20]$. This set of clusterings also gives the best result for our trials on the first dataset, in terms of compactness and separation; however, the results are not as good as those for the offline complete link algorithm with 3 and 4 clusters (corresponding to the final number of clusters given by the hybrid algorithm).

Table 6.2 shows clustering quality using sum squared error and average distance between centroids for the second datasets. In this cases, we see less sensitivity to the

TABLE 6.1

CLUSTER SUM SQUARED DEVIATION FOR DATASET 1

| $C_i$ | $C_f$ | SSQ | Avg Dist | Offline SSQ | Offline Avg Dist |
|---|---|---|---|---|---|
| 2 | 3 | $1.67 \times 10^7$ | 121 | | |
| 3 | 3 | $1.69 \times 10^7$ | 111 | $1.29 \times 10^7$ | 202 |
| 4 | 4 | $1.89 \times 10^7$ | 86.6 | $1.13 \times 10^7$ | 154 |
| 5 | 4 | $1.89 \times 10^7$ | 86.9 | | |
| 6 | 3 | $1.93 \times 10^7$ | 102 | | |
| 7 | 4 | $1.91 \times 10^7$ | 86.0 | | |
| 8 | 3 | $1.97 \times 10^7$ | 101 | | |
| $[9, 20]$ | 3 | $1.51 \times 10^7$ | 143 | | |

initial number of clusters than we did with the previous data set. $C_i \in [3, 4]$ yield identical clusterings, as do $C_i \in [5, 7]$ and $C_i \in [8, 20]$. The best clustering results from $C_i \in [3, 4]$, in terms of compactness. $C_i = 2$ outperforms $C_i \in [3, 4]$ with respect to separation, but in this case, the compactness is much worse. As with the other dataset, the hybrid algorithm is outperformed by complete link, particularly with respect to separation.

The fact that the above results show such a small range in the number of final clusters raises concerns that a few large clusters are forming early and dominating the results. To determine whether this is the case, we examine how the number of clusters change over time. Figures 6.1 and 6.2 show the call activity time series and the number of clusters over time for varying $C_i$ for the two months of data. In the first month of data (figure 6.1), the number of clusters collapses to 3 or 4 within the first 3 days after

TABLE 6.2

CLUSTER SUM SQUARED DEVIATION FOR DATASET 2

| $C_i$ | $C_f$ | SSQ | Avg Dist | Offline SSQ | Offline Avg Dist |
|---|---|---|---|---|---|
| 2 | 2 | $5.27 \times 10^7$ | 140 | | |
| 3 | 3 | $2.46 \times 10^7$ | 106 | $2.69 \times 10^7$ | 346 |
| 4 | 3 | $2.46 \times 10^7$ | 106 | $2.84 \times 10^7$ | 300 |
| 5 | 4 | $2.85 \times 10^7$ | 82.3 | | |
| 6 | 4 | $2.85 \times 10^7$ | 82.3 | | |
| 7 | 4 | $2.85 \times 10^7$ | 82.3 | | |
| $[8, 20]$ | 3 | $2.97 \times 10^7$ | 94.8 | | |

the incremental component begins. In the cases where $C_i$ is small ($< 9$), the number of clusters remains small for the remainder of the stream. In contrast, cases where $C_i \geq 9$ the number of clusters eventually settles at around 8 after about 2 weeks. It is only within the last day and a half of the time series that the number of clusters drops to around 3. In the second month of data, we do not see such a wide variation in the number of clusters; however, except for the case of $C_i = 2$, the cluster structure changes throughout the duration of the dataset.

To investigate further, we examine the root mean squared deviation (RMSD) of the clusters once the hybrid clustering algorithm has finished. In both cases, we find that there is in fact a cluster with a significantly larger RMSD than the others. Next, we look at the evolution of the RMSD over time for each of the clusters and we find that, in both cases, the large clusters appear relatively early in the execution of the algorithm. In the case of the first data set, 15% of the data items have been processed when the

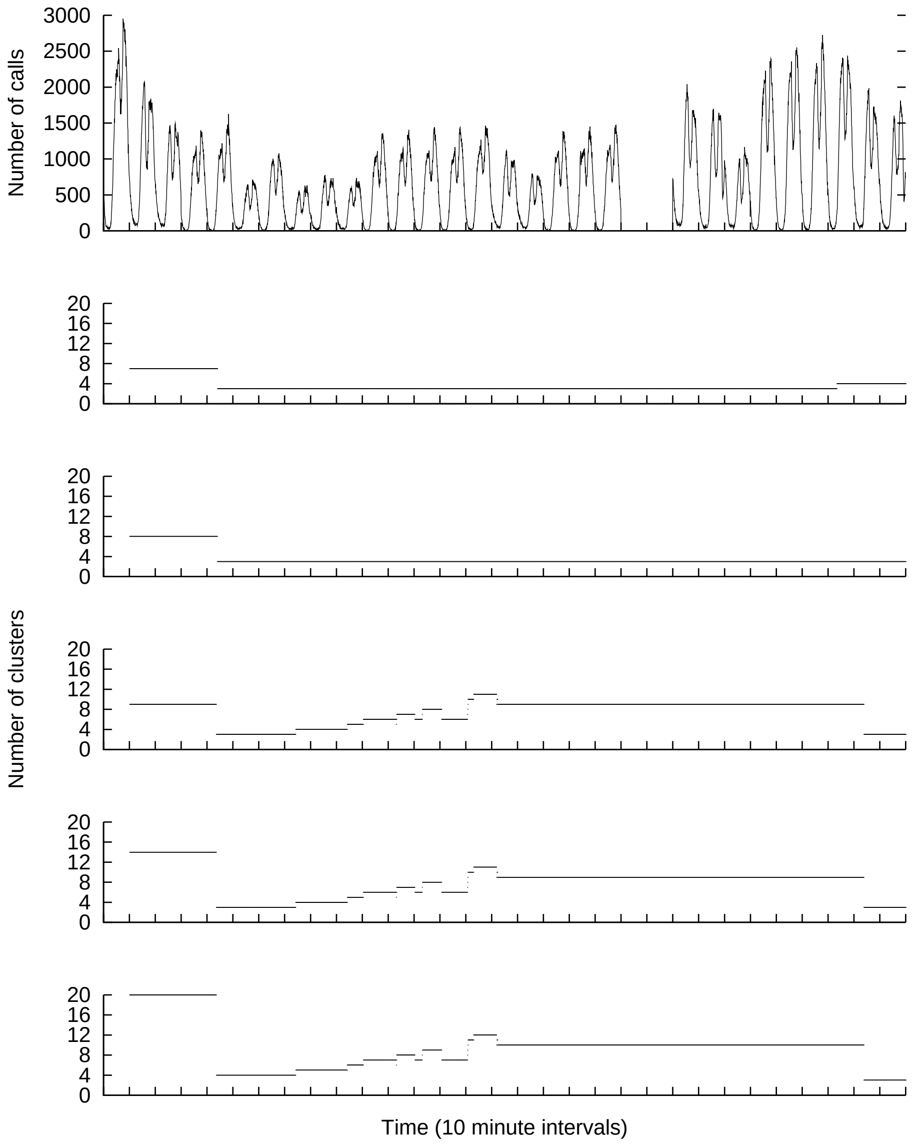


Figure 6.1. The call activity time series for the first month of data and the number of clusters over time with (from top to bottom) 7, 8, 9, 15, and 20 initial clusters.

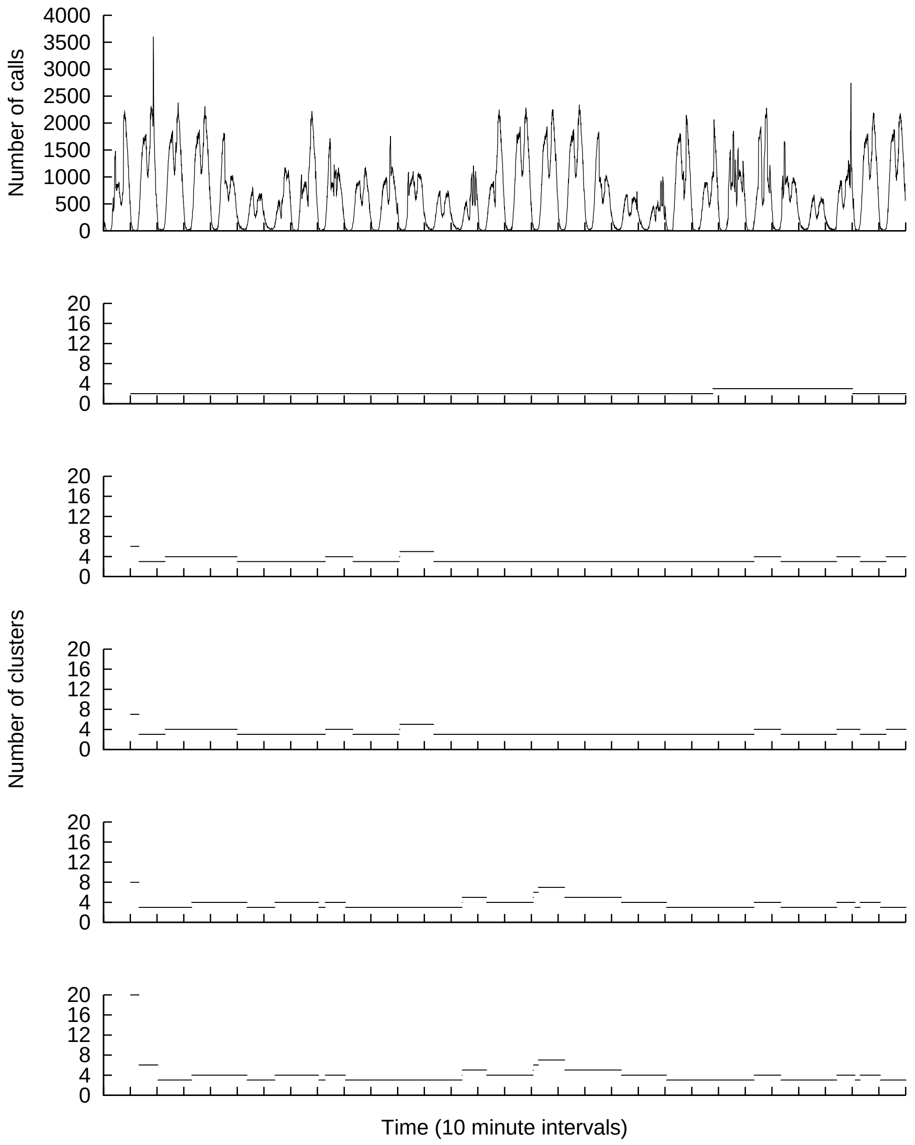


Figure 6.2. The call activity time series for the second month of data and the number of clusters over time with (from top to bottom) 2, 6, 7, 8, and 20 initial clusters.

TABLE 6.3

CLUSTER EVALUATION FOR DATASET 2 WITH MEMBERSHIP AND OVERLAP THRESHOLDS OF 3.0 AND AN INITIAL PARTITION WITH 3 CLUSTERS

| | SSQ | Avg Dist |
|---|---|---|
| Complete Link | $2.84 \times 10^7$ | 300 |
| Hybrid | $1.75 \times 10^7$ | 341 |

large cluster appears and 37% of the data items in the second dataset. We also find, however, that this cluster does not collect all of the points, as can be seen in figures 6.1 and 6.2, which both show cluster creation and merges following the point at which the large cluster appears. The fact that these large clusters appear upon a merge indicates that the large threshold we are using to determine when clusters are overlapping may be resulting in a merge of well separated clusters.

While, in general, using smaller thresholds causes the algorithm to produce a very large number of clusters, we do find one case, where both the membership and overlap thresholds are 3.0 and $C_i = 3$, that gives good results for the second dataset: the hybrid algorithm outperforms complete link with respect to both compactness and separation. In this case, the hybrid algorithm still has one cluster that dominates all others in size, and, as in the other cases, it appears early in the execution (after 19% of the data items have been processed). The quality measures for this clustering are show in table 6.3.

## 6.6 Conclusion

In this chapter, we described an online hybrid clustering algorithm for event detection that uses complete link to create initial clusters and a modification of the leader algorithm to update clusters as new data arrives. We have compared this algorithm to complete link clusterings on the full dataset using sum squared error, root mean squared deviation, and average distance between centroids with mixed results.

In the results we show, the algorithm maintains a small number of clusters relative to the number of data items in the stream; however, this result is heavily dependent on the membership threshold used to accept data items into existing clusters. If this threshold is too low, then number of clusters explodes and, in the worst case, can grow linearly with respect to the number of data items. Similarly, if the membership threshold or the merging threshold is too high, the number of clusters can collapse into a small set of large, non-descriptive clusters which give little information about the underlying data. This shortcoming is difficult to overcome since, due to the nature of the clustering features used to store the sufficient statistics of the clusters, they cannot be split. We have shown, however, that, given the proper parameterization, the algorithm can produce results better than those produced by complete link.

In the next chapter, we describe a feature clustering approach for event detection that indicates the location of the event as well as its occurrence.

# CHAPTER 7

# FEATURE CLUSTERING FOR DATA STEERING IN DYNAMIC DATA DRIVEN APPLICATION SYSTEMS

## 7.1 Abstract

In this chapter, we describe an approach for determining the location of an event that causes a dramatic increase in call activity using feature clustering on real-world data. We first examine the effect of two emergency events on the call activity in the areas surrounding the events and show how the anomalous call activity becomes dominated by normal call activity as larger areas around the event are considered. We find that, in general, the univariate models can detect the event and location more quickly, but is more prone to detecting false positives. The purpose of our system is to trigger a suite of simulations, so false positives incur a cost; therefore, we believe that the feature clustering approach we describe is better suited for our application.[1]

## 7.2 Introduction

The Wireless Phone-based Emergency Response (WIPER) system is a proof-of-concept Dynamic Data Driven Application System (DDDAS) designed to leverage real-time streaming cell phone data to provide high-level information about an emergency

[1]An earlier version of this work was published in the 2009 Proceedings of the International Conference on Computational Science [66].

situation to emergency response managers. WIPER consists of modules for automatically detecting emergency events and for running and validating predictive simulations of potential outcomes [55, 56, 67, 68]. Schoenharl and Madey [80] describe an approach for on-line simulation validation for WIPER using streaming cell phone data as it becomes available. In this paper we address the problem of identifying the area for which the simulations should be run.

In an emergency situation, it is likely that the area of interest is small relative to the total coverage area of the cell phone network. Running predictive simulations for the entire coverage area is problematic in terms of computational requirements and the amount of data produced that must in turn be validated and presented to emergency response managers. In this chapter we describe an approach for identifying the area affected by an emergency using feature clustering. We illustrate the effectiveness of this approach using two case studies of emergency events that appear in real-world cell phone data.

## 7.3 Related Work

Dynamic Data Driven Application Systems (DDDAS) are characterized by their ability to incorporate new data into running models and simulations as they become available and to steer data collection, enabling the simulations to receive and utilize the most relevant data [20, 23]. Plale *et al.* [69] use the amount of variance in an ensemble of weather forecast simulations to collect additional data and direct additional computational resources to the areas where additional simulation runs are needed. Flikkema *et al.* [28] uses data models to filter observations at the sensors. In this case, the interesting observations are those that do not match the data model, and it is these that are transmitted for further processing.

For the approach described in this chapter, the input is a data stream consisting of the time at which each call is made and the tower with which the caller's phone communicates. We transform this data into a standard machine learning data set. Let the data set **D** be an $n \times m$ matrix with $n$ data items and $m$ features. At regular time intervals, we construct a data item consisting of the number of calls made from each tower.

In this case, we are interested in analyzing how the time series for the towers, which correspond to the features of our data set, relate to each other. To accomplish this, we cluster the features, rather than the observations. This is straightforward to do, as it only requires taking the transpose of the data matrix prior to applying the clustering algorithm.

Data clustering is an unsupervised machine learning method for grouping the items of a data set **D** based on some distance measure. Hierarchical algorithms identify a nested set of partitions in the data. Most hierarchical methods take an agglomerative approach, meaning that there are initially $n$ clusters, each containing one data item in **D**. These clusters are iteratively merged until all of the data items belong to the same cluster. Popular agglomerative clustering algorithms include single-link and complete-link. To illustrate these two clustering methods, consider a graph where the data items are represented as vertices and edges are added between two vertices in increasing order of distance between the two corresponding data items. At each step, the clusters in the single-link approach are the connected components and the clusters in the complete-link approach are the completely connected components [47].

Rodrigues *et al.* [74] describe an algorithm for clustering the features of a data stream. The algorithm is a divisive-agglomerative algorithm that uses a dissimilarity measure based on correlation along with a Hoeffding bound to determine when clusters are split. The algorithm relies on the fact that the pairwise correlation of the time series, $corr(\vec{a}, \vec{b})$, can be computed using a small number of sufficient statistics. For each time series it is necessary to keep track of $\sum_{i=1}^{n} a_i$, $\sum_{i=1}^{n} b_i$, $\sum_{i=1}^{n} a_i^2$, and $\sum_{i=1}^{n} b_i^2$, and for each pair of time series $\sum_{i=1}^{n} a_i b_i$ must be updated with the arrival of each data item. Additionally, the number of data items that have arrived so far, $n$, must be known. Rodrigues *et al.* [74] use correlation distance, $diss(\vec{a}, \vec{b}) = 1 - corr(\vec{a}, \vec{b})$, as a dissimilarity measure.

An alternative approach, described by [10], applies a univariate model at each location and uses methods from percolation theory to determine the impact of an event. We examine two univariate models: one is simply the $z$-scores for the time series and the other uses Holt-Winters forecasting.

The $z$-score gives the deviation of a point from the mean in units of standard deviation:

$$z_{\mu,\sigma} = \frac{x - \mu}{\sigma} \tag{7.1}$$

The $z$-score is commonly used in statistical process control to identify mechanical problems [7].

There are two aspects of distributions relevant to the $z$-score: location and dispersion. Most commonly, mean, $\mu$, and standard deviation, $\sigma$, are used; however, both are sensitive to noise which, unfortunately, is present in our datasets. In our particular case, we have missing records which, when aggregated into call counts, skews the distribution to the left.

As the number of outliers in a dataset increases, detecting outliers tends to become more difficult because the outliers begin to affect the data model. There are a few approaches to dealing with this problem, and we briefly examine two: cleaning the data and using robust outlier detection techniques. Liu *et al* [54] describe an on-line approach for cleaning outliers in time series data using Kalman filters. This approach replaces outlying points according to the filter. Alternatively, Chaing *et al* [16] describes several robust outlier detection that can tolerate noise in some fraction of the data points (up to 50% in some cases). We opt for the latter approach.

The median, $\tilde{\mu}$ provides a robust measure of location. Assuming that the distribution is symmetric, and by using the $z$-score we are, $\mu = \tilde{\mu}$ [9]. Rousseeuw and Croux [75] describe a scale estimator, $Q_n$, that is robust when up to 50% of the data points are random noise and efficiently estimates the standard deviation for Gaussian distributions. The scale estimator is the $\binom{n}{2}/4$ order statistic of the interpoint distances in the population:

$$Q_n = d\left\{\left|x_i - x_j\right|; i < j\right\}_{(\binom{n}{2}/4)} \tag{7.2}$$

where $d = 2.2219$ is a constant that scales the values of $Q_n$ to correspond with those of $\sigma$.

Using the robust location estimator median, $\tilde{\mu}$, and scale estimator $Q_n$, we define a robust $z$-score, $z_{\tilde{\mu},Q_n}$:

$$z_{\tilde{\mu},Q_n} = \frac{x - \tilde{\mu}}{Q_n} \tag{7.3}$$

We denote the standard $z$-score $z_{\mu,\sigma}$.

The second univariate model, Holt-Winters forecasting, estimates the value at a particular time step as the weighted average of previous observations. Let $L_t$ be the observed value, or level, at time $t$. The expected level, $\tilde{L}_t$, is the weighted sum of the current observed value, $L_t$ and the forecasted value at the previous time step $\tilde{L}_{t-1}$. Let $\alpha \in [0, 1]$ be the weight, or smoothing coefficient. Then

$$\tilde{L}_t = \alpha L_t + (1 - \alpha)\tilde{L}_{t-1}. \tag{7.4}$$

This model can be expanded to account for seasonality by introducing a seasonality factor, $I_t$ to the estimate of $\tilde{L}_t$. Let $p$ be the frequency of the seasonality, *e.g.* if the period is one year and data is collected monthly, $p = 12$. The estimate of $\tilde{L}_t$ with multiplicative seasonality becomes

$$\tilde{L}_t = \alpha \frac{L_t}{I_{t-p}} + (1-\alpha)\tilde{L}_{t-1}. \tag{7.5}$$

The seasonality factor is a weighted sum of the current ratio of observed level to the forecast level and the past ratios for the particular time interval, where $\delta \in [0, 1]$ is the smoothing coefficient for the seasonality:

$$\tilde{I}_t = \delta \frac{L_t}{\tilde{L}_t} + (1-\delta)I_{t-p} \tag{7.6}$$

[13, 93].

Alternatively, the seasonality can be additive, in which case the estimate of the level becomes:

$$\tilde{L}_t = \alpha(L_t - I_{t-p}) + (1-\alpha)\tilde{L}_{t-1}. \tag{7.7}$$

and the seasonality index is updated by

$$\tilde{I}_t = \delta(L_t - \tilde{L}_t) + (1-\delta)I_{t-p}. \tag{7.8}$$

Finally, we add an additional component for the trend for multiplicative seasonality

$$\tilde{L}_t = \alpha \frac{L_t}{I_{t-p}} + (1-\alpha)(\tilde{L}_{t-1} + T_{t-1}). \tag{7.9}$$

and additive seasonality

$$\tilde{L}_t = \alpha(L_t - I_{t-p}) + (1 - \alpha)\tilde{L}_{t-1} \tag{7.10}$$

The trend component is the weighed sum of the change in expected level from the last time step and the previous trend value:

$$T_t = \gamma(\tilde{L}_t - \tilde{L}_{t-1}) + (1 - \gamma)T_{t-1}. \tag{7.11}$$

In our analysis, we use the additive model because our time series contain intervals where no calls are made, meaning the value at that time is 0. As a consequence, the multiplicative seasonality is undefined. Therefore, we compute the level, seasonal index, and trend as follows:

$$\tilde{L}_t = \alpha(L_t - I_{t-p}) + (1 - \alpha)\tilde{L}_{t-1}, \tag{7.12}$$

$$\tilde{I}_t = \delta(L_t - \tilde{L}_t) + (1 - \delta)I_{t-p}, \tag{7.13}$$

and

$$T_t = \gamma(\tilde{L}_t - \tilde{L}_{t-1}) + (1 - \gamma)T_{t-1}. \tag{7.14}$$

We use the R implementation of Holt-Winters forecasting, which determines the initial starting points for the level and trend using linear regression. The initial values of the seasonal index are obtained by finding the average of the levels, $\overline{L}$ over the first period and setting $I_i = L_i - \overline{L}$ for $i \in [1, p]$ [70].

We use basic percolation theory to combine the results obtained from the univariate models, following the approach described by Candia *et al* [10]. Percolation theory is concerned with the connected components that arise on a lattice where sites, vertices, or bonds, edges, are activated at random, creating a set of subgraphs [32, 82]. Candia *et al* [10] showed the properties of percolating graphs generated from normal data differ from those generated from data containing a large, anomalous event.

We use an underlying graph built from the locations in the datasets rather than using a lattice. The locations are represented as vertices and are connected with an edge if they are neighbors. For the postal codes, we extract the edges using a PostGIS query that returns pairs of postal codes that are "touching." The coverage area of the towers is typically approximated using a Voronoi lattice [10, 34, 78, 91], so we compute the Delaunay triangulation, where an edge is present between two towers if their Voronoi cells share a common edge, to obtain the edges for the tower graph [5].

## 7.4 Experimental Setup

We examine the expression of two events in real-world cell phone usage data. The first event is an explosion, and the second is a large gathering, related to a sporting event. We identify the approximate time and location (latitude and longitude) of the events using news reports and maps of the city[2].

Our data set consists of call records that give the time at which a call is made and the originating tower of each phone call. We aggregate this data to generate a time series for each tower that gives the number of calls made in successive 10 minute intervals. In some cases, we spatially aggregate these time series to obtain the total call activity over 10 minute intervals for all towers in a particular area, *e.g.* a postal code.

To explore the effect of events on the call activity in the surrounding area, we use PostGIS functions to identify the towers within a particular distance of the estimated location for the relevant time frame and aggregate the tower time series described above

[2]The events are drawn from real-world data from a particular city. Due to a non-disclosure agreement, we can not name the city or give specifics of the events.

to produce a single time series for the area.

We make heavy use of *z*-scores throughout this chapter. In computing these *z*-score, we take into account the fact that the values of interest will vary based on the day of the week and the time of day. Since we are using 10 minute intervals, a week consists of 1008 intervals. We compute the measures of location and scale for each of these intervals and use these to compute the relevant *z*-score at each time step. We also use the error in Holt-Winters forecasting, $e_{HW}$, which is the deviation of the predicted value from the actual value.

## 7.5 Results

We first look at how the events affect the call volume in the surrounding areas. The columns in figure 7.1 show the time series of call activities for the five days leading up to each event. Each row, from the top of the figure to the bottom, includes data from a larger area surrounding the location of the events. The first event (left column) occurs in the morning on the fifth day, and we see a corresponding increase in call activity at this time. The severity of this spike in activity decreases as the radius of the area increases from 1 km to 5 km. The second event (right column) occurs very early in the morning the fifth day, though we see elevated call activity even before midnight. As with the first scenario, the spike in call activity becomes less dramatic as a larger area, up to 2 km in radius, surrounding the event is included.

Next we look at the pairwise distances between the time series of the affected postal code and its neighbors. Figures 7.2 and 7.3 each show the correlation (both cumulative and over a sliding window) and Euclidean distances between the postal codes in which the events occur and the neighboring postal codes for two weeks leading up to the events. The left columns show the cumulative correlation distance used by Rodrigues

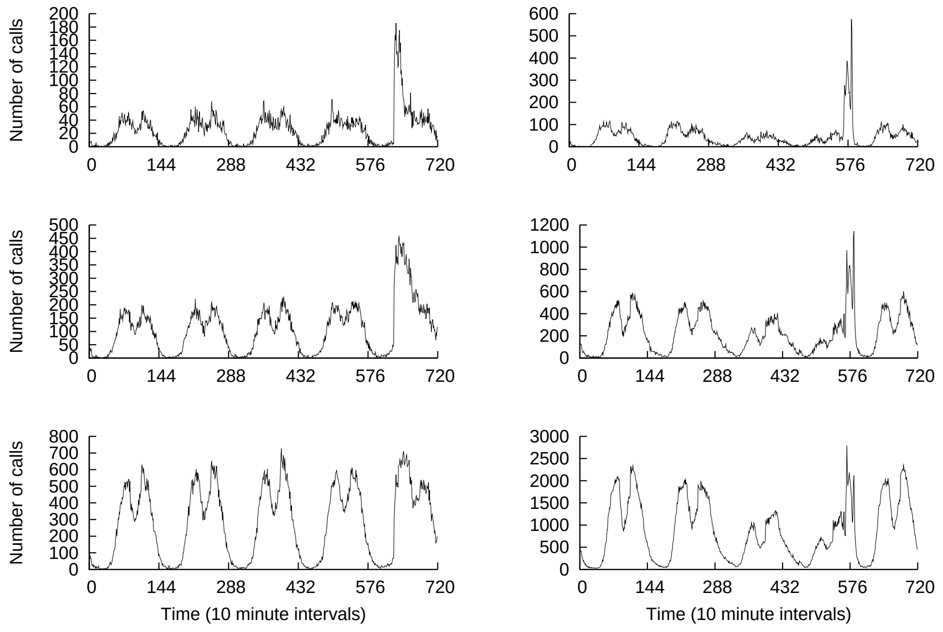


Figure 7.1. The effect of the event situations on the call activity through the surrounding cell towers. The left column shows the time series for the first situation, which occurs at during the morning of the fifth day in the time series. The right column shows the time series for the second situation, which occurs very early in the morning on the fifth day. In both cases, the severity of the activity spike decreases as a greater area is considered.

*et al.* [74]. The center and right columns show the correlation distance and Euclidean distance, respectively, over a one day sliding window.

In figure 7.2, we see an increase in each distance measure at the end of each time series. The cumulative correlation distance has only a slight increase at the end of the time series when the event happens. These increases are more dramatic in the cases where a sliding window is used. In the time series in figure 7.2 there are two days of missing data, from 576 to 864 minutes. These missing data are not noticeable in the cumulative correlation distance; however, they lead to undefined correlation distances and Euclidean distances of 0 for 144 time steps when the entire sliding window contains

0 for all features. In figure 7.3 we see similar increases in distance in most cases. The fact that the cumulative correlation distance shows only a small increase compared to the case where only a portion of the history of the time series is considered may indicate that this distance measure is dominated by older observations, making this cumulative measure insensitive to anomalies. The detrimental affect of old, stale data is discussed by Aggarwal in [2].

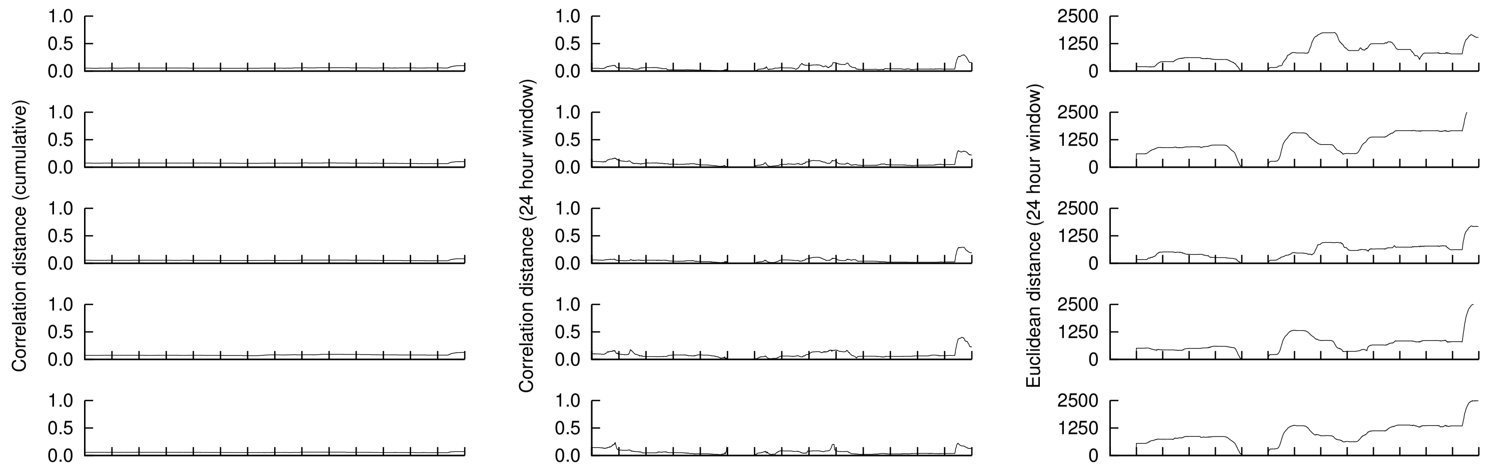
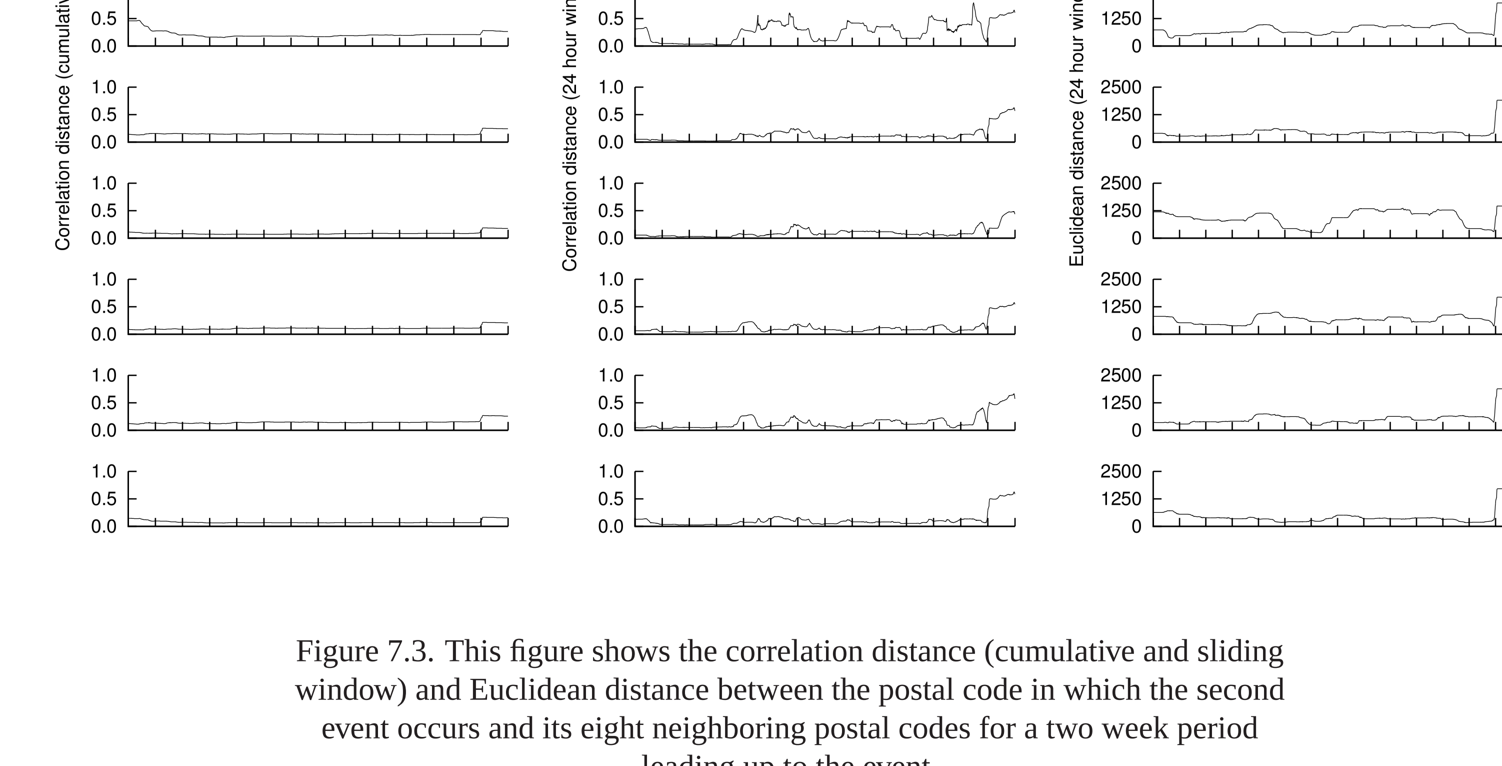


Figure 7.2. This figure shows the correlation distance (cumulative and sliding window) and Euclidean distance between the postal code in which the first event occurs and its five neighboring postal codes for a two week period leading up to the event.

Figure 7.3. This figure shows the correlation distance (cumulative and sliding window) and Euclidean distance between the postal code in which the second event occurs and its eight neighboring postal codes for a two week period leading up to the event.

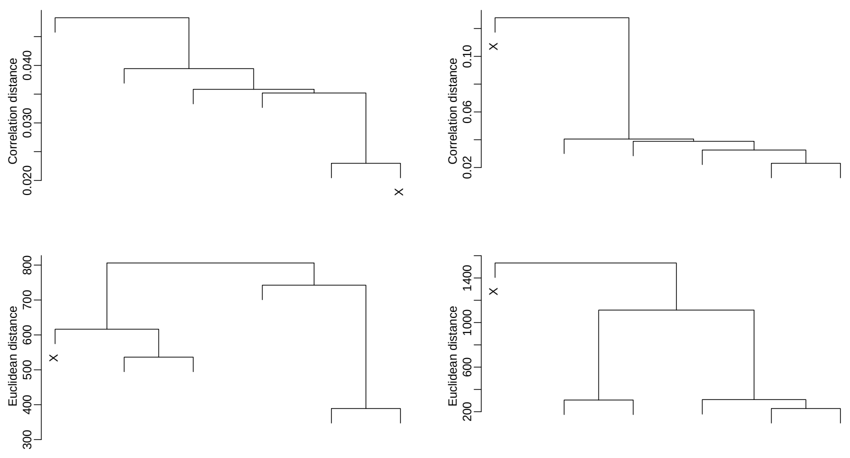


In figures 7.4 and 7.5, we compare single link clustering dendrograms for a day of normal activity and the day of the event. We cluster each day of data with the single link agglomerative algorithm using two different dissimilarity measures: correlation distance and Euclidean distance. Figure 7.4 shows the clusters for the first emergency situation. In both the correlation and Euclidean distance clusterings, the distance, the postal code in which the emergency occurred is significantly larger than the distance between any two clusters on the day of normal activity. In figure 7.5, we see a similar separation of the postal code in which the emergency occurred along with one neighbor from the remaining clusters, though the increase in distance is not as dramatic as in the previous case. It is not surprising that the outlying cluster on the day of the emergency contains two postal codes since there is one postal code, shown by the time series in the top row of figure 7.3, that does not exhibit an the same increase in distance seen in the others, indicating that the call activity in that area is also affected by the event.

Applying what we see in these figures, we compute the maximum intercluster distance for single link clusterings over a sliding window. That is, we run single link until there are two clusters and compute the distance between them. To put these values in perspective with respect to the larger dataset, we compute the $z$-score at each time step.

Figure 7.6 shows the $z$-scores for the time around the first event (the bombing) using the postal code level data. The left panel shows the standard $z$-score, $z_{\mu,\sigma}$, and the right shows the robust $z$-score, $z_{\tilde{\mu},Q_n}$. In both cases, the call activity increases in the hour after the event. This dataset is particularly affected by noise, and by using the more robust measures of location, $\tilde{\mu}$, and scale, $Q_n$, the inactive time leading up to the event, while still high, is closer to 0 without affecting the increase in $z$-score associated with the event.

Figure 7.7 shows the $z$-scores for the time around the first emergency using tower

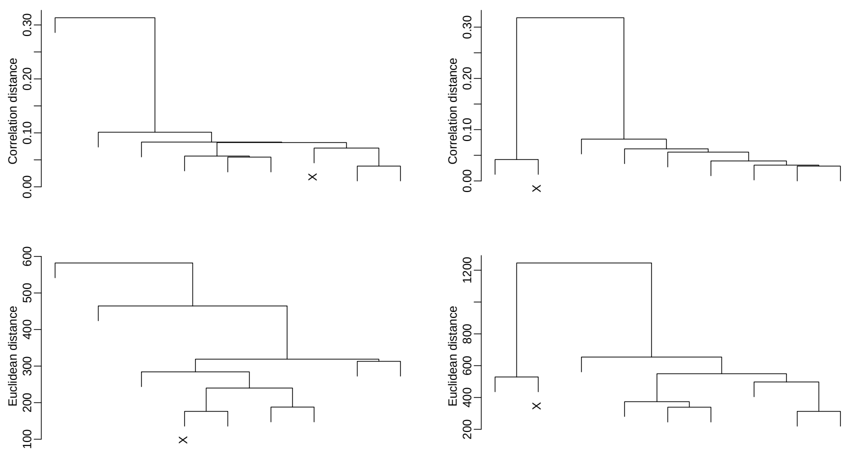


(a) Normal (b) Emergency

Figure 7.4. The clustering of the call volume time series for postal codes surrounding the first emergency event. The left column shows the clustering of one day of normal activity and the right column shows the clustering of the day of the first emergency event. The leaf representing the postal code in which the event occurs is marked with an X. For both the correlation distance (top row) and Euclidean distance (bottom row), the affected postal code is near other clusters during the day of normal activity but is an outlier during the day of the event.

level data. The left panel shows $z_{\mu,\sigma}$ and the right shows $z_{\tilde{\mu},Q_n}$. At this level of granularity, we are unable to detect the event. In both plots, we see an increase approximately two hours before the event; however, the ranges of $z$-scores are very small indicating that the increase is not substantial.

Figures 7.8 and 7.9 show the $z$-scores for the time around the second event at the postal code and tower levels, respectively. At different levels of granularity, we detect different aspects of this event. At the postal code level, we detect the activities toward the end of the event, at time 2, after the crowd has gathered with the team. At the tower levels, we detect activities toward the beginning of the event, around time 0 when the sporting event ends.



(a) Normal (b) Emergency

Figure 7.5. The clustering of the call volume time series for postal codes surrounding the second emergency event. The left column shows the clustering of one day of normal activity and the right column shows the clustering of the day of the second emergency event. The leaf representing the postal code in which the event occurs is marked with an X. In this case, the call activity in one of the neighboring postal codes is also affected by this event, resulting in an outlying cluster with to data items.

We also use the maximum cluster distance from complete link clustering. Since the distance between two complete link clusters is the maximum intercluster distance, we only need to compute the diameter of the set of points in the dataset. Figures 7.10 and 7.11 show the *z*-scores for the diameter of the data points at the postal code and tower levels, respectively. At the postal code level, we do see increases for both the standard *z*-scores; however, they only appear a few hours after the event. This makes sense in light of the fact that the maximum cluster distance for the complete link clusters are larger than those for single link. This measure is, therefore, less sensitive to a small number of anomalies in one of the features in the sliding window, so more anomalies must accumulate before there is an impact on the measure. As with the single link clustering, we do not detect the event at the postal code level.

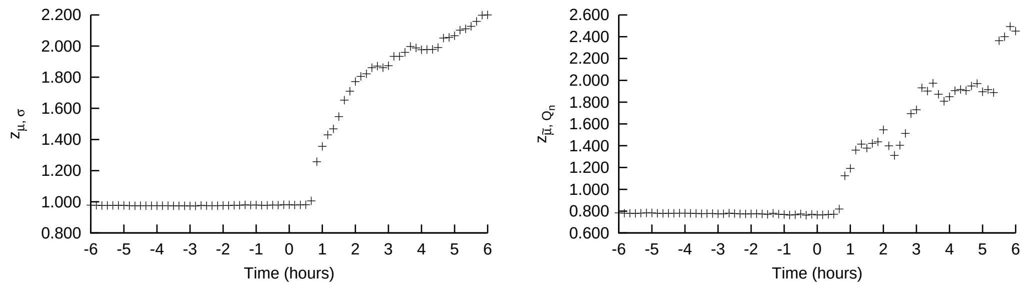


Figure 7.6. The *z*-scores of the maximum distance between single link clusters of the postal code time series over a 12 hour window surrounding the first event. The standard *z*-score, $z_{\mu,\sigma}$ appears on the left, and the robust *z*-score, $z_{\tilde{\mu},Q_n}$, appears on the right.

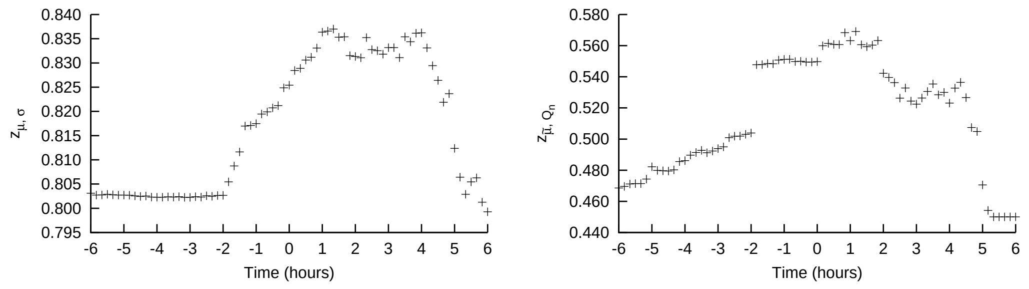


Figure 7.7. The *z*-scores of the maximum distance between single link clusters of the tower time series over a 12 hour window surrounding the first event. The standard *z*-score, $z_{\mu,\sigma}$ appears on the left, and the robust *z*-score, $z_{\tilde{\mu},Q_n}$, appears on the right.

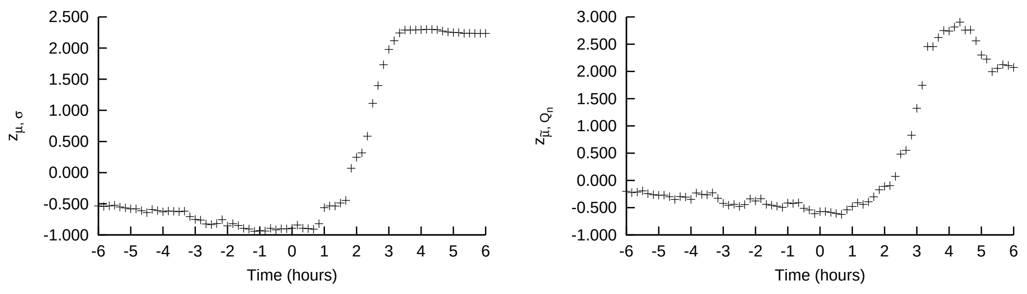


Figure 7.8. The *z*-scores of the maximum distance between single link clusters of the postal code time series over a 12 hour window surrounding the second event. The standard *z*-score, $z_{\mu,\sigma}$ appears on the left, and the robust *z*-score, $z_{\tilde{\mu},Q_n}$, appears on the right.

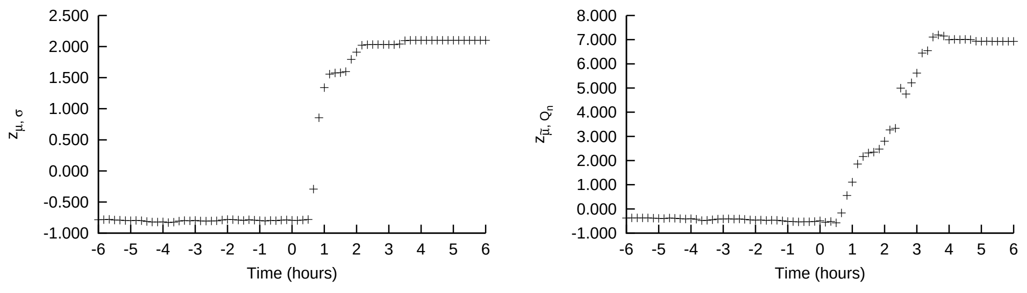


Figure 7.9. The *z*-scores of the maximum distance between single link clusters of the tower time series over a 12 hour window surrounding the second event. The standard *z*-score, $z_{\mu,\sigma}$ appears on the left, and the robust *z*-score, $z_{\tilde{\mu},Q_n}$, appears on the right.

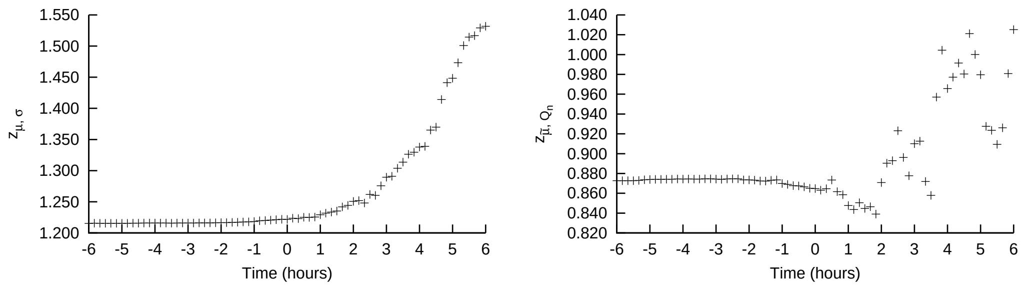


Figure 7.10. The *z*-scores of the maximum distance between complete link clusters of the postal code time series over a 12 hour window surrounding the first event. The standard *z*-score, $z_{\mu,\sigma}$ appears on the left, and the robust *z*-score, $z_{\tilde{\mu},Q_n}$, appears on the right.

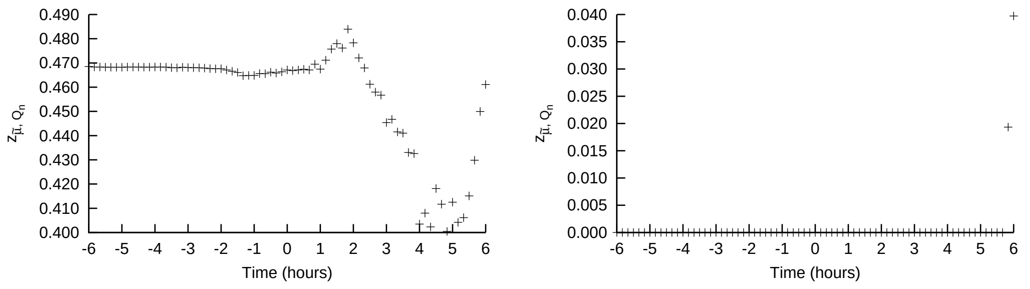


Figure 7.11. The *z*-scores of the maximum distance between complete link clusters of the tower time series over a 12 hour window surrounding the first event. The standard *z*-score, $z_{\mu,\sigma}$ appears on the left, and the robust *z*-score, $z_{\tilde{\mu},Q_n}$, appears on the right.

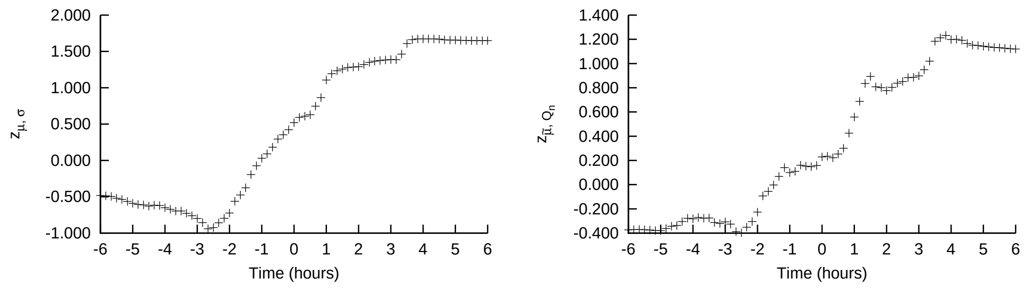


Figure 7.12. The *z*-scores of the maximum distance between complete link clusters of the postal code time series over a 12 hour window surrounding the second event. The standard *z*-score, $z_{\mu,\sigma}$ appears on the left, and the robust *z*-score, $z_{\tilde{\mu},Q_n}$, appears on the right.

Figures 7.12 and 7.13 show the *z*-scores for the diameter of the data points for the second event at the postal code and tower levels, respectively. As with the single link clustering, the two levels of granularity detect different aspects of the event. The *z*-scores jump at the end of the event when using postal code level data and at the beginning of the event when using tower level data.

Next, we explore using a univariate model at each location. We use two different models: the Holt-Winters model where we measure the difference between the value we expect, based on the model, at each time step and the actual value, and a Gaussian model where we measure the $z_{\tilde{\mu},Q_n}$-score.

One way we can use this model to detect events is to simply look for outliers at each location, and when outliers are found, the location with the greatest deviation from the expected value is assumed to be the location of the event. For each time step surrounding the two events, we find the location with the greatest deviation from the expected value and compute its distance from the event. When working with postal code level data, we use the distance between the postal code centroids; otherwise, we use the distance between the towers. Figure 7.14 shows these distances for the first

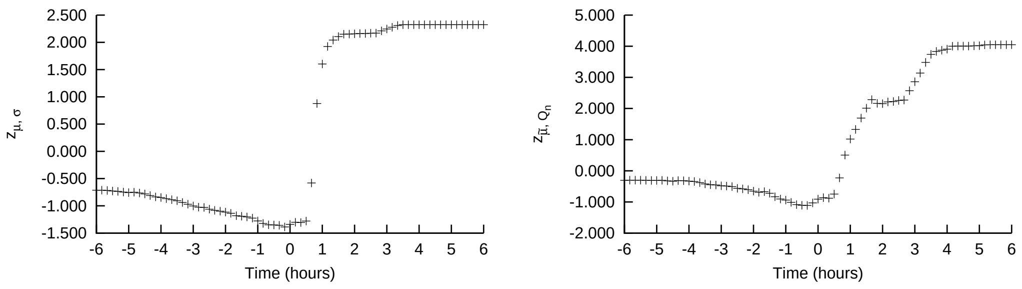


Figure 7.13. The *z*-scores of the maximum distance between complete link clusters of the tower time series over a 12 hour window surrounding the second event. The standard *z*-score, $z_{\mu,\sigma}$ appears on the left, and the robust *z*-score, $z_{\tilde{\mu},Q_n}$, appears on the right.

event (top panel) at the postal code (left) and tower (right) levels. In both cases, we find the event, with the distance dropping to zero and $e_{HW}$ increasing suddenly shortly after time 0.

The results are not as clear with the second event, shown in figure 7.15. There is a spike in $e_H W$ around time 1, but there is no corresponding decrease in the distance. This is likely due to the fact this event involves a migration of the population from the stadium to the square (we are measuring the distances from the latter).

Figures 7.16 and 7.17 show the results for the Gaussian model. As with the Holt-Winters model, the first event can easily be seen at time 0 where the distance drops to 0 and max($z_{\tilde{\mu},Q_n}$) increases sharply. Additionally, this effect is sustained more so than with the Holt-Winters approach, probably because this model, at each time step, doesn't depend on data smoothed at each previous time step. For the second event, we see a decrease in the distance between times 1 and 2, which corresponds to the gathering of the crowd. Note that in this case, the event takes place near a boundary between two postal codes, which is why we see small changes in the distance in the top left after time 1. The spikes in max($z_{\tilde{\mu},Q_n}$) do not correspond to the decreases in distance, but we

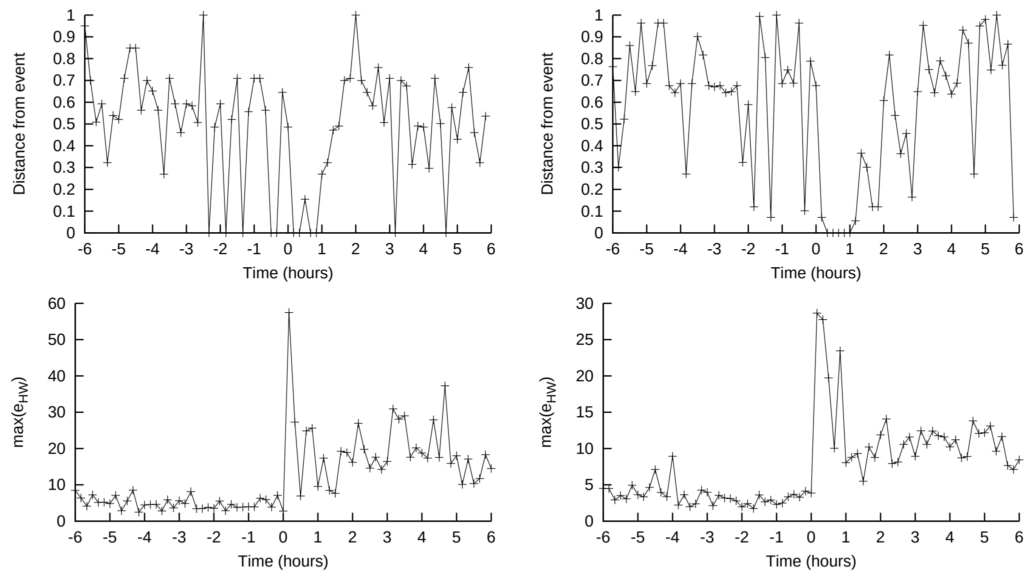


Figure 7.14. The normalized distance of the location with the maximum $e_{HW}$ from the location of the event (postal codes on the left and towers on the right) at each time step in the 12 hours surrounding the first event.

can attribute this to the fact that this data is produced by a large crowd migrating from one location to another.

Another approach for detecting events using the univariate models is to find clusters of anomalous regions using percolation tools as described in Candia *et al* [10]. In this approach, we build a graph that represents locations as nodes, with edges between two nodes if they are neighbors, *i.e.* postal codes or Voronoi cells that share a boundary, and both are anomalous, based on some *z*-score threshold at each time step. Once we have this graph, we identify the largest component as the potential event location. Figures 7.18 through 7.25 shows the largest component sizes and how this component size compares to other largest component sizes for the same day of the week and time of day using the *z*-score. At no point in these figures does a largest component size deviate significantly from the expected value. These events are significantly smaller

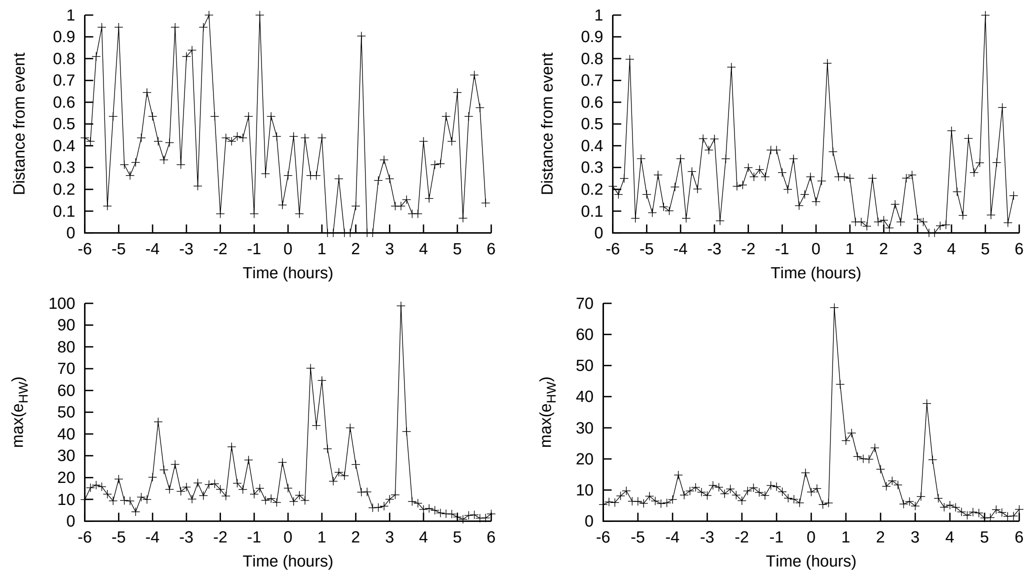


Figure 7.15. The normalized distance of the location with the maximum $e_{HW}$ from the location of the event (postal codes on the left and towers on the right) at each time step in the 12 hours surrounding the second event.

than the one analyzed in [10], and we are unable to detect them because they do not affect a large enough area to generate a large connected component in the percolation graph.

## 7.6 Discussion and Conclusions

In this chapter, we have examined the impact of two events on the call activity in the areas surrounding the events. We have analyzed the relevant data using feature clustering, a basic Gaussian univariate model, and Holt-Winters forecasting.

The feature clustering approach has the advantage of providing a single *z*-score for identifying whether the call activity is in an anomalous state or not; however there are some costs associated with this. First, the clustering method does not detect the event as quickly as the univariate methods due to the sliding window. The first anomalous data

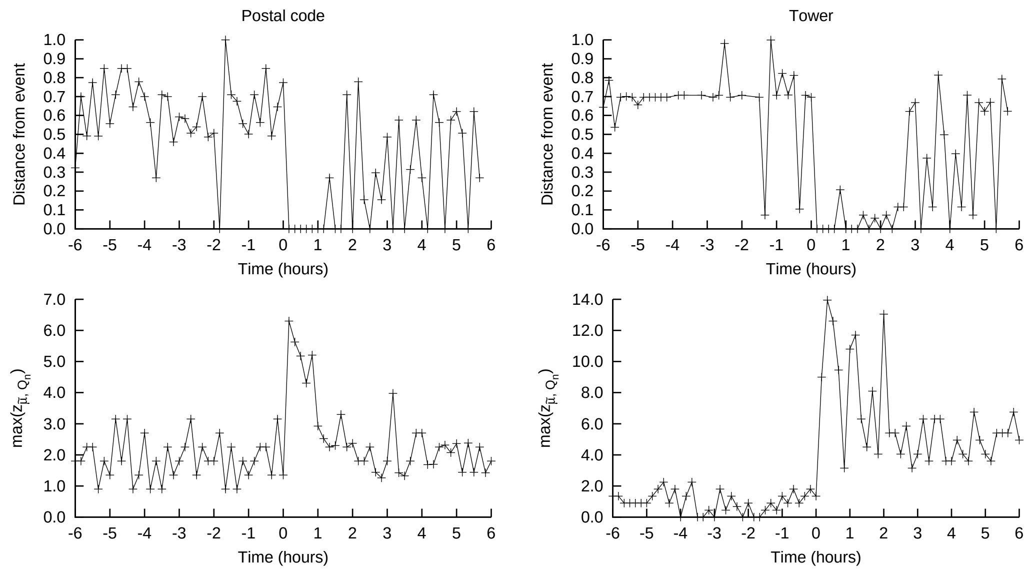


Figure 7.16. The top panel shows the normalized distance of the location with the maximum $z_{\tilde{\mu},Q_n}$-score from the location of the event (postal codes on the left and towers on the right) at each time step in the 12 hours surrounding the first event. The bottom panel shows the maximum $z_{\tilde{\mu},Q_n}$-score at each time step.

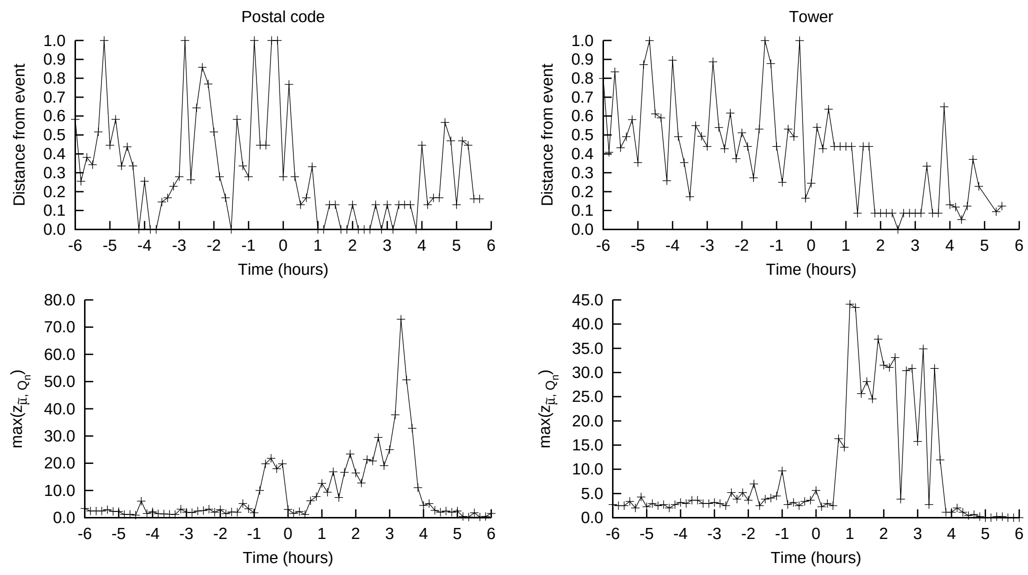


Figure 7.17. The top panel shows the normalized distance of the location with the maximum $z_{\tilde{\mu},Q_n}$-score from the location of the event (postal codes on the left and towers on the right) at each time step in the 12 hours surrounding the second event. The bottom panel shows the maximum $z_{\tilde{\mu},Q_n}$-score at each time step.

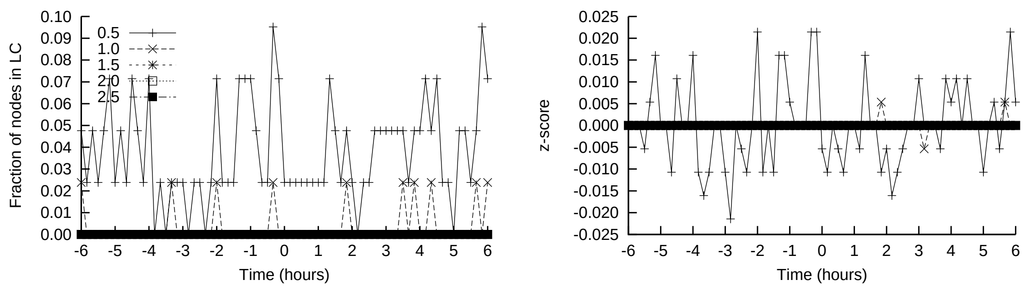


Figure 7.18. The fraction of nodes, representing postal codes, in the largest component of the percolating graphs where the sites are activated when the $z_{\mu,\sigma}$-score of the difference of the forecasted call activity from the actual call activity is $\{0.5, 1.0, 1.5, 2.0, 2.5\}$ (left) and the $z_{\mu,\sigma}$-score of the largest component size, taking the day of the week and time of day effects into account (right) for the first event for the 12 hours surrounding that event.

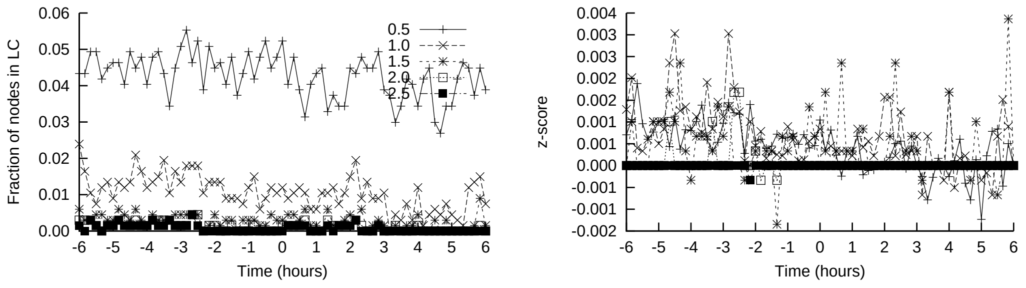


Figure 7.19. The fraction of nodes, representing towers, in the largest component of the percolating graphs where the sites are activated when the $z_{\mu,\sigma}$-score of the difference of the forecasted call activity from the actual call activity is $\{0.5, 1.0, 1.5, 2.0, 2.5\}$ (left) and the $z_{\mu,\sigma}$-score of the largest component size, taking the day of the week and time of day effects into account (right) for the first event for the 12 hours surrounding that event.

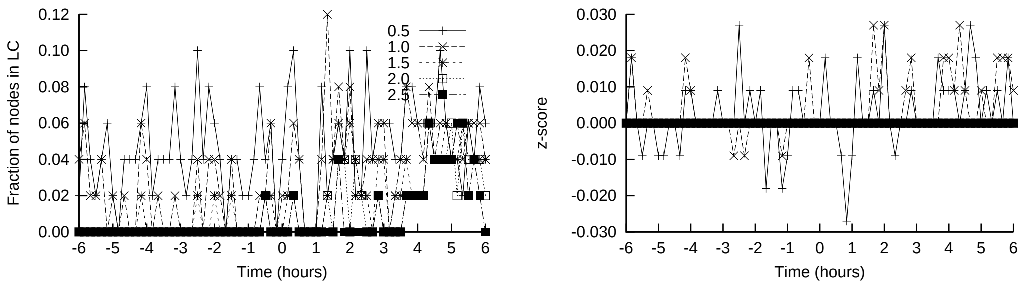
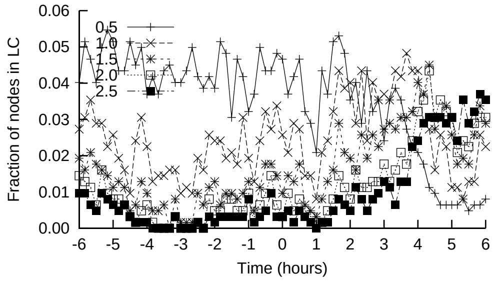


Figure 7.20. The fraction of nodes, representing postal codes, in the largest component of the percolating graphs where the sites are activated when the $z_{\mu,\sigma}$-score of the difference of the forecasted call activity from the actual call activity is $\{0.5, 1.0, 1.5, 2.0, 2.5\}$ (left) and the $z_{\mu,\sigma}$-score of the largest component size, taking the day of the week and time of day effects into account (right) for the second event for the 12 hours surrounding that event.



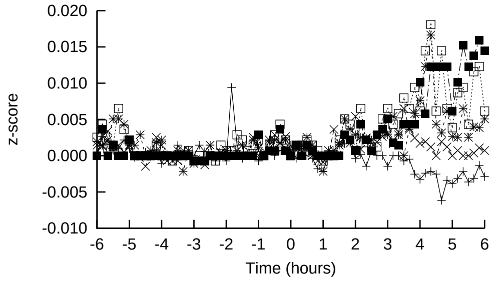
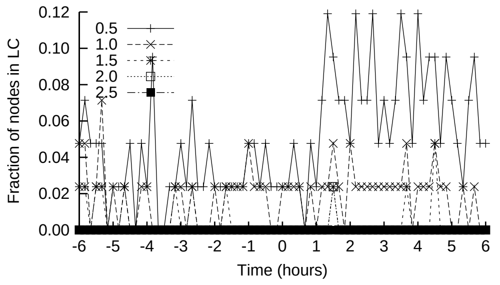


Figure 7.21. The fraction of nodes, representing towers, in the largest component of the percolating graphs where the sites are activated when the $z_{\mu,\sigma}$-score of the difference of the forecasted call activity from the actual call activity is $\{0.5, 1.0, 1.5, 2.0, 2.5\}$ (left) and the $z_{\mu,\sigma}$-score of the largest component size, taking the day of the week and time of day effects into account (right) for the second event for the 12 hours surrounding that event.



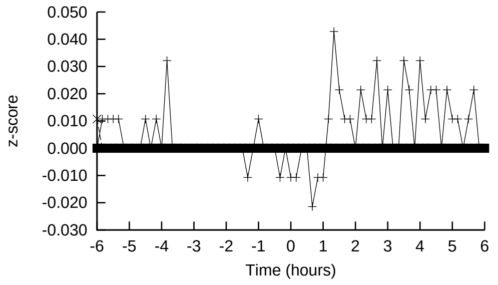


Figure 7.22. The fractions of node, representing postal codes, in the largest component of the percolating graph where sites are activated at $z_{\tilde{\mu},Q_n} = \{0.5, 1.0, 1.5, 2.0, 2.5\}$ (left) and the $z_{\mu,\sigma}$-score of the largest component size, taking the day of the week and time of day effects into account (right) for the first event in the 12 hours surrounding that event.

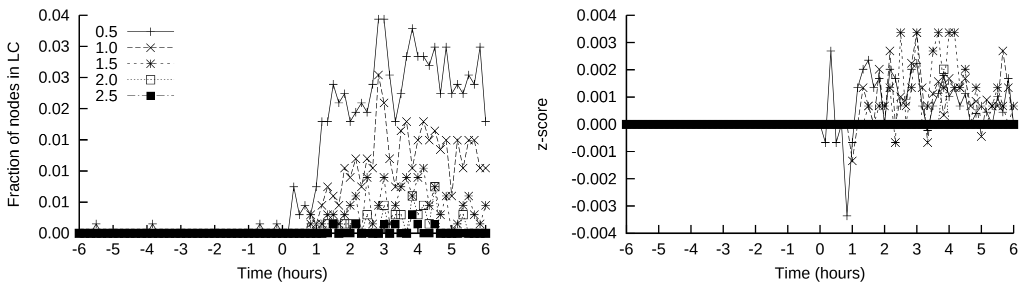


Figure 7.23. The fractions of nodes, representing towers, in the largest component of the percolating graph where sites are activated at $z_{\tilde{\mu},Q_n} = \{0.5, 1.0, 1.5, 2.0, 2.5\}$ (left) and the $z_{\mu,\sigma}$-score of the largest component size, taking the day of the week and time of day effects into account (right) for the first event in the 12 hours surrounding that event.

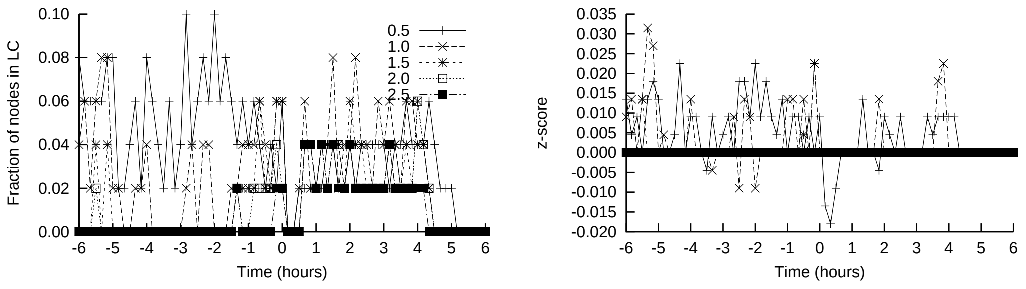


Figure 7.24. The fractions of node, representing postal codes, in the largest component of the percolating graph where sites are activated at $z_{\tilde{\mu},Q_n} = \{0.5, 1.0, 1.5, 2.0, 2.5\}$ (left) and the $z_{\mu,\sigma}$-score of the largest component size, taking the day of the week and time of day effects into account (right) for the second event in the 12 hours surrounding that event.

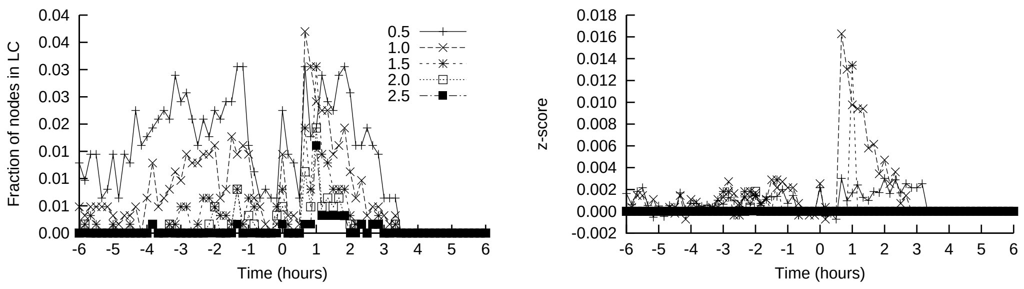


Figure 7.25. The fractions of nodes, representing towers, in the largest component of the percolating graph where sites are activated at $z_{\tilde{\mu},Q_n} = \{0.5, 1.0, 1.5, 2.0, 2.5\}$ (left) and the $z_{\mu,\sigma}$-score of the largest component size, taking the day of the week and time of day effects into account (right) for the first event in the 12 hours surrounding that event.

point will likely not have a major impact on the clustering because the affected time series will be dominated by normal values; it may take several anomalous data points before the clustering is affected. As a result, this approach will not be able to detect short events that do not generate enough anomalous data points to affect the clustering. Similarly, once an event is detected, it will take time for the anomalous data points to work their way out of the sliding window, meaning that the *z*-score will indicate an anomaly after the event has subsided. This can also cause issues in the case where the cause of the increase in call activity is moving. It is likely that only the initial location of the event would be detected by the feature clustering approach.

While the feature clustering approach has several disadvantages in general, it seems to be a good option for our application: automatically starting predictive simulations. The types of events that we are interested in simulating will persist over longer periods of time, so it is not desirable to detect short events. In fact, we may consider the detection of such events to be false positives, since starting simulations for them serves no purpose and there is a cost associated with starting a simulation suite. Additionally, for the same reason, this approach is resistant to isolated (spatially and temporally) noise that would be identified as events using a simple threshold with the univariate models. Since the purpose of the system is only to start the simulations, this is not an issue. Once the simulations are started, they are responsible for gathering more information as needed. The main drawback that remains is the inability to detect a second event while the first remains in the sliding window. It may be possible to address this issue by ignoring the time series in the most outlying cluster until the data has been evicted from the window.

# CHAPTER 8

# SUMMARY

In this dissertation, we have described research performed as part of the development of the Wireless Phone-Based Emergency Response System. We have described a data warehouse that serves as a historical data source for the system as well as for more general research in the topics of complex networks and human mobility. We have also described a clustering algorithm for the detection and alert system that identifies potential emergency events in space and time.

The warehouse is designed to facilitate efficient data extract for researchers with little experience in using relational database systems, allowing them to devote most of their attention to their research rather than making sense of the organization of the data. We also identify and discuss an issue associated with the merge of the call records: the introduction of duplicate records due to clock skew in the CDR data.

We describe two clustering algorithms, an online algorithm that, given the correct parameters, produces a model of the data comparable to offline methods, and a feature clustering algorithm that detects events causing a significant and localized increase in call activity. The feature clustering algorithm is suitable for triggering the simulation and prediction component of the WIPER system as well as giving the system guidance on which spatial area is of greatest interest, fulfilling the data steering aspect of the dynamic data driven application system paradigm.

## 8.1 Contributions

This dissertation makes three notable contributions: first, we present an extension of the dimensional model for the case where there is a great deal of missing dimensional data. This model follows naturally from a partitioning of the fact table and allows us to include foreign key constraints where necessary without adding a large number of empty records to the dimensional tables. In our case, we partition the data based on whether caller or recipients of voice calls or SMS has an account with a particular service provider, but this approach could also be applied to areas such as biological networks where data is, at this point, still incomplete and being collected experimentally.

Next, we describe an approach for merging noisy, redundant data with very little supplementary data available to aid in record matching. There are often a number of identifying fields available for comparison when merging two databases; however, in our case, we only have the primary key: a timestamp, the caller ID, and the recipient ID. The problem is further complicated by the fact that there may be two records with the same caller ID and recipient ID in quick succession, so looking for small deviations in the timestamp is not sufficient. We use an approach that merges records that match in caller ID and recipient ID in increasing order of deviation in timestamp, and we use a feature of the data, the intercall time distribution, to determine the stopping point.

Finally, we present a feature clustering approach for detecting events and their location in real world phone data. We compare this approach to simple univariate models at each location and find that while this approach takes longer to identify the event, it is less susceptible to false negatives, which incur a cost in our application.

As evidence of these contributions, we list the presentations and publications that have come out of this work.

### 8.1.1 Presentations

The work in this dissertation has yielded a several of scholarly presentations:

- “Anomaly Detection in a Mobile Communication Network.” North American Association for Computational Social and Organization Sciences. 2006. Notre Dame, IN.
- “WIPER: An Emergency Response System.” International Community on Information Systems for Crisis Response and Management. 2008. Washington, D.C.
- “Feature Clustering for Data Steering in Dynamic Data Driven Application Systems.” Dynamic Data Driven Application Systems Workshop. International Conference on Computational Science. 2009. Baton Rouge, LA.

### 8.1.2 Publications

The work has also yielded a number of scholarly publications:

- Alec Pawling, Ping Yan, Julián Candia, Tim Schoenharl, and Greg Madey. “Anomaly Detection in Streaming Sensor Data,” In: *Intelligent Techniques for Warehousing and Mining Sensor Network Data.* Alfredo Cuzzocrea, Ed. IGI Global. Forthcoming.
- Alec Pawling and Greg Madey. “Feature Clustering for Data Steering in Dynamic Data Driven Application Systems.” Computational Science - ICCS 2009: 9th International Conference, Baton Rouge, USA, Proceedings, Part II, Gabrielle Allen, Jaroslaw Nabrinsky, Edward Seidel, Geert Dick van Albada, Jack J. Dongarra, and Peter M. A. Sloot (eds), Lecture Notes in Computer Science series, vol 5545, Springer-Verlag, Heidelberg, 2009.

- Alec Pawling, Tim Schoenharl, Ping Yan, and Greg Madey. “WIPER: An Emergency Response System.” In F. Fiedrich and B. V. de Walle, ed. Proceedings of the 5th International ISCRAM Conference, 2008.

- Alec Pawling, Nitesh Chawla, and Greg Madey. “Anomaly Detection in a Mobile Communication Network.” Computational & Mathematical Organization Theory. 13:4, December, 2007.

- Alec Pawling, Nitesh Chawla, and Greg Madey. “Anomaly Detection in a Mobile Communication Network.” In: Proceedings of the Annual Conference of the North American Association for Computational Social and Organization Sciences. 2006. (Received Best Student Paper Award).

## 8.2 Limitations and Future Work

The work we have presented here with respect to the historical data source provides the foundation for a more comprehensive system in the future. Currently, the warehouse is an extract machine that allows the researchers to easily and efficiently retrieve relevant call records; however, no analysis tools are provided. In describing the data cleaning process, we present a method for eliminating redundancy resulting from merging call records from different sources; however, we do not identify the stopping point of this process. It should also be noted that only a high level of data cleaning has been applied, and additional cleaning specific to a particular type of analysis may be required before that analysis can be effectively performed.

The stream clustering algorithm we present can produce good clusterings of the data, but is highly sensitive to the threshold values. We also do not address the issue of retiring stale clusters due to the fact our datasets used in the development of this algorithm cover a relatively short period of time.

The feature clustering algorithm is limited by the fact that once an event occurs, it will be present in the sliding window for its duration. This makes discovering smaller events that happen after a large event difficult to detect. This problem may be mitigated by the simulation and prediction system. Once the initial event has been detected and the simulations have been started, the changes in call activity that are due to secondary events, assuming that they are in the area over which the simulations are being run, will be incorporated into the simulations as they are validated and updated.

# BIBLIOGRAPHY

1. Sloan Digital Sky Survey / SkyServer. `http://cas.sdss.org/dr7/en`. Accessed July 7, 2009.

2. C. C. Aggarwal. On biased reservoir sampling in the presence of stream evolution. In *Proceedings of the 32nd Conference on Very Large Databases*, 2006.

3. C. C. Aggarwal, J. Han, J. Wang, and P. S. Yu. A framework for clustering evolving data streams. In *Proceedings of the 29th Conference on Very Large Data Bases*, Berlin, Germany, 2003. VLDB Endowment.

4. G. Allen. Building a dynamic data driven applicatoin system for hurricane prediction. In Y. Shi, G. D. van Albada, J. Dongarra, and P. M. A. Sloot, editors, *Proceedings of the International Conference on Computational Science*, volume 4487 of *Lecture Notes in Computer Science*, pages 988–994, Berlin Heidelberg, 2007. Springer-Verlag.

5. F. Aurenhammer. Voronoi diagrams—a survey of a fundamental geometric data structure. *ACM Computing Surveys*, 23(3):345–405, 1991.

6. J. Becla and D. L. Wang. Learned from managing a petabyte. In *Proceedings of the 2005 Conference on Innovative Data Systems Research*, 2005.

7. C. Bicking and F. M. Gryna, Jr. *Quality Control Handbook*, chapter Process Control by Statistical Methods, pages 23–1—23–35. McGraw Hill, New York, NY, USA, 1979.

8. P. Bradely, U. Fayyad, and C. Reina. Scaling clustering algorithms to large databases. In *Proceedings of the 4th International Conference on Knowledge Discovery and Data Mining*, 1998.

9. M. G. Bulmer. *Principles of Statistics*. Dover, 1979.

10. J. Candia, M. González, P. Wang, T. Schoenharl, G. Madey, and A.-L. Barabási. Uncovering individual and collective human dynamics from mobile phone records. *Journal of Physics A: Mathematical and Theoretical*, 41:224015, 2008.

11. B. Carbunar, A. Grama, J. Vitek, and O. Carbunar. Redundancy and coverage detection in sensor networks. *ACM Transactions on Sensor Networks*, 2(1):94–128, February 2006.

12. M. Charikar, C. Chekuri, T. Feder, and R. Motwani. Incremental clustering and dynamic information retrieval. In *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, pages 626–635. ACM Press, 1997.

13. C. Chatfield and M. Yarvis. Holt-winter forecasting: Some practical issues. *The Statistician*, 37:129–140, 1988.

14. A. R. Chaturvedi, S. A. Filatyev, J. P. Gore, A. Hanna, J. Means, and A. K. Mellema. Integrating fire, structure and agent models. In V. S. Sunderam, G. D. v Albada, P. M. A. Sloot, and J. Dongarra, editors, *Proceedings of the International Conference on Computational Science*, volume 3515 of *Lecture Notes in Computer Science*, pages 695–702, Berlin, Germany, 2005. Springer-Verlag.

15. E. Y. Cheu, C. Keongg, and Z. Zhou. On the two-level hybrid clustering algorithm. In *International Conference on Artificial Intelligence in Science and Technology*, pages 138–142, 2004.

16. L. H. Chiang, R. J. Pell, and M. B. Seasholtz. Exploring process data with the use of robust outlier detection algorithms. *Journal of Process Control*, 13:437–449, 2003.

17. H. Chipman and R. Tibshirani. Hybrid hierarchical clustering with applications to microarray data. *Biostatistics*, 7(2):286–301, 2006.

18. E. Codd. A relational model of data for large shared data banks. *Communications of the ACM*, 13(6):377–387, June 1970.

19. T. M. Cover and J. A. Thomas. *Elements of Information Theory*. John Wiley & Sons, 2nd edition, 2006.

20. F. Darema. Dynamic data driven applications systems: A new paradigm for application simulations and measurements. In *Proceedings of the ICCS 2004, Lecture Notes in Computer Science 3038,* 2004.

21. S. B. Davidson, J. Crabtree, B. P. Brunk, J. Schug, V. Tannen, G. C. Overton, and J. C. J. Stoeckert. K2/Kleisli and GUS: Experiments in integrated access to genomic data sources. *IBM Systems Journal*, 40:512–530, 2001.

22. C. Douglas and A. Deshmukh. Dynamic data driven application systems. http://www.dddas.org/NSFworkshop2000.html, March 2000.

23. C. C. Douglas. DDDAS: Dynamic Data Driven Application Systems. http://www.dddas.org, 2008.

24. K. Eilbeck, A. Brass, N. Paton, and C. Hodgman. INTERACT: an object oriented protein-protein interaction database. In *Prceedings of the International Conference on Intelligent Systems for Molecular Biology*, 1999.

25. R. Elmasri and S. B. Navathe. *Fundamentals of Database Systems*. Addison Wesley, 2000.

26. F. Farnstrom, J. Lewis, and C. Elkan. Scalability for clustering algorithms revisited. *SIGKDD Explorations*, 2:51–57, 2000.

27. I. P. Fellegi and A. B. Sunter. A theory for record linkage. *Journal of the American Statistical Association*, 64(328):1183–1210, 1969.

28. P. G. Flikkema, P. K. Agarwal, J. S. Clark, C. Ellis, A. Gelfand, K. Munagala, and J. Yang. From data reverence to data relevance: Model-mediated wireless sensing of the physical environment. In Y. Shi, G. D. van Albada, J. Dongarra, and P. M. A. Sloot, editors, *Proceedings of the International Conference on Computational Science*, volume 4487 of *Lecture Notes in Computer Science*, pages 988–994, Berlin Heidelberg, 2007. Springer-Verlag.

29. B. Fortner. HDF: The hierarchical data format. *Dr. Dobb's Journal*, 23(5):42, 1998.

30. R. Fujimoto, R. Guensler, M. Hunter, H.-K. Kim, J. Lee, J. Leonard II, M. Palekar, K. Schwan, and B. Seshasayee. Dynamic data driven application simulation of surface transportation systems. In V. N. Alexandrov, G. D. val Albada, P. M. A. Sloot, and J. Dongarra, editors, *Proceedings of the International Conference on Computational Science*, volume 3993 of *Lecture Notes in Computer Science*, pages 425–432, Berlin, Germany, 2006. Springer-Verlag.

31. M. Gaynor, M. Seltzer, S. Moulton, and J. Freedman. A dynamic, data-driven decision support system for emergency medical services. In V. S. Sunderam, G. D. v Albada, and P. M. A. S. ad J. Dongarra, editors, *Proceedings of the International Conference on Computational Science*, volume 3515 of *Lecture Notes in Computer Science*, pages 703–711, Berlin, Germany, 2005. Springer-Verlag.

32. G. Gimmett. *Percolation*, volume 321 of *A Series of Comprehensive Studies in Mathematics*. Springer-Verlag, 2nd edition, 1999.

33. M. C. González and A.-L. Barabási. Complex networks: From data to models. *Nature Physics*, 3:224–225, April 2007.

34. M. C. González, C. A. Hidalgo, and A.-L. Barabási. Understanding individual human mobility patterns. *Nature*, 453:779–782, 2008.

35. A. D. Gordon. *Classification*. Chipman & Hall, 2nd edition, 1999.

36. J. Gray, D. T. Liu, M. Nieto-Santisteban, A. Szalay, D. J. DeWitt, and G. Herber. Scientific data management in the coming decade. *SIGMOD Record*, 34(4):34–41, 2005.

37. B. Gruschke. Integrated event management: Event correlation using dependency graphs. In *Proceedings of the 9th IFIP/IEEE International Workshop on Distributed Systems Operation & Management*, 1998.

38. S. Guha, A. Meyerson, N. Mishra, R. Motwani, and L. O'Callaghan. Clustering data streams: Theory and practice. *IEEE Transactions on Knowledge and Data Engineering*, 3:515–528, May/June 2003.

39. M. Halkidi, Y. Batistakis, and M. Vazirgiannis. On clustering validation techniques. *Journal of Intelligent Information Systems*, 17(2/3):107–145, 2001.

40. J. Handl, J. Knowles, and D. B. Kell. Computational cluster validation in post-genomic data analysis. *Bioinformatics*, 21(15):3201–3212, 2005.

41. J. A. Hartigan. *Clustering Algorithms*. Wiley Series in Probability and Mathematical Statistics. John Wiley & Sons, New York, NY, USA, 1975.

42. M. A. Hernández and S. J. Stolfo. Real-world data is dirty: Data cleansing and the merge/purge problem. *Data Mining and Knowledge Discovery*, 2(1):1573, 1998.

43. P. E. Hodges, W. E. Payne, and J. I. Garrels. The yeast protein database (ypd): A curated proteome database for *saccharomyces cerevisiae*. *Nucleic Acids Research*, 26(1):68–72, 1998.

44. W. Inmon. *Building the Data Warehouse*. John Wiley & Son, 2nd edition, 1996.

45. W. H. Inmon. *Building the Data Warehouse*. John Wiley & Son, 4th edition, 2005.

46. A. K. Jain and R. C. Dubes. *Algorithms for Clustering Data*. Prentice Hall, 1988.

47. A. K. Jain, M. N. Murty, and P. J. Flynn. Data clustering: A review. *ACM Computing Surveys*, 31(3):264–323, September 1999.

48. A. Kasprzyk, D. Keefe, D. Smedley, D. London, W. Spooner, C. Melsopp, M. Hammond, P. Rocca-Serra, T. Cox, and E. Birney. EnsMart: A generic system for fast and flexible access to biological data. *Genome Research*, 14:160–169, 2004.

49. L. Kaufman and P. J. Rousseeuw. *Finding Groups in Data: An Introduction to Cluster Analysis*. John Wiley & Son, 1990.

50. R. Kimball and M. Ross. *The Data Warehouse Toolkit*. John Wiley & Son, 2002.

51. S. Kullback and R. A. Leibler. On information and sufficiency. *The Annals of Mathematical Statistics*, 22:79–86, 1951.

52. J. Küntzer, C. Backes, T. Blum, A. Gerasch, M. Kuarmann, O. Kohlbacher, and H.-P. Lenhof. BNDB – the biochemical network database. *BMC Bioinformatics*, 8:367, 2007.

53. T. J. Lee, Y. Pouliot, V. Wagner, P. Gupta, D. W. Stringer-Calvert, J. D. Tenenbaum, and P. D. Karp. BioWarhouse: a bioinformatics database warehouse toolkit. *BMC Bioinformatics*, 7:170, 2006.

54. H. Liu, S. Shah, and W. Jiang. On-line outlier detection and data cleaning. *Computers and Chemical Engineering*, 28:1635–1647, 2004.

55. G. Madey. WIPER: The Integrated Wireless Phone-based Emergency Response System. http://www.nd.edu/˜dddas, 2008.

56. G. R. Madey, A.-L. Barabási, N. V. Chawla, M. Gonzalez, D. Hachen, B. Lantz, A. Pawling, T. Schoenharl, G. Szabó, P. Wang, and P. Yan. Enhanced situational awareness: Application of DDDAS concepts to emergency and disaster management. In Y. Shi, G. D. van Albada, J. Dongarra, and P. M. A. Sloot, editors, *ICCS 2007*, volume 4487 of *LNCS*, pages 1090–1097, Heidelberg, 2007. Springer.

57. G. R. Madey, G. Szabó, and A.-L. Barabási. WIPER: The integrated wireless phone based emergency response system. In V. N. Alexandrov, G. D. val Albada, P. M. A. Sloot, and J. Dongarra, editors, *Proceedings of the International Conference on Computational Science*, volume 3993 of *Lecture Notes in Computer Science*, pages 417–424, Berlin, Germany, 2006. Springer-Verlag.

58. J. Mandel, J. D. Beezley, L. S. Bennethum, S. Chakraborty, J. L. Coen, C. C. Douglas, J. Hatcher, M. Kim, and A. Vodacek. A dynamic data driven wildland fire model. In Y. Shi, G. D. van Albada, J. J. Dongarra, and P. M. A. Sloot, editors, *Proceedings of the International Conference on Computational Science*, volume 4487 of *Lecture Notes in Computer Science*, pages 1042–1049, Heidelberg, Germany, 2007. Springer-Verlag.

59. F. J. Massey. The kolmogorov-smirnov test for goodness of fit. *American Statistical Association Journal*, 46(253):68–78, March 1951.

60. B. McMillan. The basic theorems of information theory. *The Annals of Mathematical Statistics*, 24(2):196–219, June 1953.

61. H. W. Mewes, K. Heumann, A. Kaps, K. Mayer, F. Pfeiffer, S. Stocker, and D. Frishman. MIPS a database for genomes and protein sequences. *Nucleic Acids Research*, 27(1):44–48, 1999.

62. W. O'Mullane, N. Li, M. Nieto-Santisteban, A. Szalay, and A. Thakar. Batch is back: Casjobs, serving multi-tb data on the web. In *Proceedings of the IEEE International Conference on Web Services*, 2005.

63. J.-P. Onnela, J. Saramäki, J. Hyvövnen, G. Szabó, D. Lazer, K. Kaski, J. Kertész, and A.-L. Barabási. Structure and tie strengths in mobile communication networks. *Proceedings of the National Academy of Sciences*, 104(18):7332–7336, 2007.

64. A. Pawling, N. V. Chawla, and G. Madey. Anomaly detection in a mobile communication network. In *Proceedings of the North American Association for Computational Social and Organization Science*, 2006.

65. A. Pawling, N. V. Chawla, and G. Madey. Anomaly detection in a mobile communication network. *Computational & Mathematical Organization Theory*, 13(4):407–422, December 2007.

66. A. Pawling and G. Madey. Feature sclustering for data steering in dynamic data driven application systems. In G. Allen, J. Nabrizyski, E. Seidel, G. D. van Albada, J. J. Dongarra, and P. M. A. Sloot, editors, *Proceedings of the 9th International Conference on Computational Science.*, volume 5545 of *Lecture Notes on Computer Science*, pages 460–469, Heidelberg, May 2009. Springer-Verlag.

67. A. Pawling, T. Schoenharl, P. Yan, and G. Madey. WIPER: An emergency response system. In F. Fiedrich and B. V. de Walle, editors, *Proceedings of the 5th International ISCRAM Conference*, 2008.

68. A. Pawling, P. Yan, J. Candia, T. Schoenharl, and G. Madey. *Intelligent Techniques for Warehousing and Mining Sensor Network Data*, chapter Anomaly Detection in Streaming Sensor Data. IGI Global, 2009.

69. B. Plale, D. Gannon, D. Reed, S. Graves, K. Droegemeier, B. Wihelmson, and M. Ramamurthy. Towards dynamically adaptive weather analysis and forecasting in LEAD. In V. Sunderam, B. van Albada, P. Sloot, and J. Dongarra, editors, *Proceedings of the International Conference on Computational Science*, volume 3515 of *Lecture Notes in Computer Science*, pages 624–631, Berlin Heidelberg, 2005. Springer-Verlag.

70. R Development Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2010. ISBN 3-900051-07-0.

71. L. Ramakrishnan, Y. Simmhan, and B. Plale. Realization of dynamically adaptive weather analysis and forecasting in LEAD: Four years down the road. In Y. Shi, G. D. van Albada, J. J. Dongarra, and P. M. A. Sloot, editors, *Proceedings of the International Conference on Computational Science*, volume 4487 of *Lecture Notes*

*in Computer Science*, pages 1122–1129, Heidelberg, Germany, 2007. Springer-Verlag.

72. R. Ramakrishnan and J. Gehrke. *Database Management Systems*. McGraw-Hill, 2003.

73. R. Rew and G. Davis. NetCDF: An interface for scientific data access. *IEEE Computer Graphics & Applications*, 10(4):76–82, 1990.

74. P. Rodrigues, J. Gama, and J. P. Pedroso. Hierarchical time-series clustering for data-streams. In *Proceedings of the First International Workshop on Knowledge Discovery in Data Streams*, 2004.

75. P. J. Rousseeuw and C. Croux. Alternatives to the median absolute deviation. *Journal of the American Statistical Association*, 88(424):1273–1283, December 1993.

76. S. M. Savaresi, D. L. Boley, S. Bittanti, and G. Gazzaniga. Cluster selection in divisive clustering algorithms. In *Proceedings of the 2nd SIAM ICDM*, 2002.

77. T. Schoenharl. *Updating and Validating Simulations in a Dynamic Data Driven Application System*. PhD thesis, University of Notre Dame, 2007.

78. T. Schoenharl, R. Bravo, and G. Madey. WIPER: Leveraging the cell phone network for emergency response. *International Journal of Intelligent Control and Systems*, 11(4):209–216, 2006.

79. T. Schoenharl, G. Madey, G. Szabó, and A.-L. Barabási. WIPER: A multi-agent system for emergency response. In B. van de Walle and M. Turoff, editors, *Proceedings of the 3rd International Information Systems for Crisis Response and Management Conference*, 2006.

80. T. W. Schoenharl and G. Madey. Evaluation of measurement techniques for the validation of agent-based simulations against streaming data. In M. Bubak, G. D. van Albada, J. Dongarra, and P. M. Sloot, editors, *Computational Science – ICCS 2008*, volume 5103 of *LNCS*, pages 6–15. Springer, 2008.

81. C. E. Shannon. A mathematical theory of communication. *The Bell System Technical Journal of Artificial Intelligence Research*, 27:379–423, 1948.

82. D. Stauffer and A. Aharony. *Introduction to Percolation Theory*. Taylor & Francis, second edition, 1994.

83. M. Steinder and A. S. Sethi. The present and future of event correlation: A need for end-to-end service fault localization. In *Proceedings of 2001 Multi-Conference on Systemics, Cybernetics, and Informatics*, 2001.

84. R. Sunderraman. *Oracle 8 Programming: A Primer*. Addison Wesley, 2000.

85. M. Surdeanu, J. Turmo, and A. Ageno. A hybrid unsupervised approach for document clustering. In *Proceedings of the 5th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2005.

86. B. Tatomir and L. Rothkrantz. Crisis management using mobile ad-hoc wireless networks. In *Proceedings of the Second International ISCRAM Conference*, April 2005.

87. The PostgreSQL Global Development Group. PostgreSQL 8.3.3 documentation. http://www.postgresql.org/files/documentation/pdf/8.3/postgresql-8.3-US.pdf. Accessed 3 April 2009., 2008.

88. M. Thomas, F. Andoh-Baidoo, and S. George. EVResponse - moving beyond traditional emergency response notification. In *Proceedings of the Eleventh Americas Conference on Information Systems*, 2005.

89. S. Tilak, N. B. Abu-Ghazaleh, and W. Heizelman. Collaborative storage management in sensor networks. *International Journal of Ad Hoc and Ubiquitous Computing*, 1(1/2):47–58, 2005.

90. M. C. Vuran, O. B. Akan, and I. F. Akyildiz. Spatio-temporal correlation: Theory and applications for wireless sensor networks. *Computer Networks*, 45:245–259, 2004.

91. P. Wang, M. González, C. Hidalgo, and A.-L. Barabási. Understanding the spreading patterns of mobile phone viruses. *Science*, 324:1071–1076, 2009.

92. D. C. Wells, E. W. Greisen, and R. H. Harten. FITS: A flexible image transport system. *Astronomy & Astrophyics Supplement Series*, 44:363–370, 1981.

93. P. R. Winters. Forecasting sales by exponentially weighted moving averages. *Management Science*, 6(3):324–342, 1960.

94. I. H. Witten and E. Frank. *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufman, 2nd edition, 2005.

95. T. Zhang, R. Ramakrishnan, and M. Livny. BIRCH: an efficient data clustering method for very large databases. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 103–114, June 1996.

*This document was prepared & typeset with LaTeX $2_\varepsilon$, and formatted with* NDdiss$2_\varepsilon$ *classfile (v3.0[2005/07/27]) provided by Sameer Vijay.*