This Dissertation Proposal

entitled

A SYSTEM FOR DETECTING ANOMALIES IN DATA STREAMS

FOR EMERGENCY RESPONSE APPLICATIONS

typeset with NDdiss2_ε v3.0 (2005/07/27) on September 10, 2007 for

Alec Pawling

A SYSTEM FOR DETECTING ANOMALIES IN DATA STREAMS FOR EMERGENCY RESPONSE APPLICATIONS

A Dissertation Proposal

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, B.S., M.S.

Greg Madey, Director

Graduate Program in Computer Science and Engineering

Notre Dame, Indiana

September 2007

A SYSTEM FOR DETECTING ANOMALIES IN DATA STREAMS FOR EMERGENCY RESPONSE APPLICATIONS

Abstract

by

Alec Pawling

This document proposes research for a Ph.D. dissertation in Computer Science and Engineering. We intend to develop two components of an emergency response system: a Detection and Alert System and a Real-Time Data Source.

The Detection and Alert System will identify anomalies in a stream of telecommunication data that may indicate an emergency has occurred or an interesting situation has arisen in a developing situation. We will consider three components of the telecommunication data when looking for anomalies: the call volumes, the geographical distribution of calls, and the social networks formed by the call transactions.

The Real-Time Data Source will receive a raw stream of telecommunication transactions and provide temporal and spatially aggregated summaries of the data to the Detection and Alert System, as well as other components of the emergency response system. In addition to timely distribution of the summarized data to the clients, we also require synchronization between all components of the WIPER system.

ACKNOWLEDGMENTS

This material is based upon work supported by the National Science Foundation under Grant No. CNS-0540348.

CONTENTS

FIGURES

TABLES

CHAPTER 1

INTRODUCTION

This document proposal research on two problems: (1) fast anomaly detection in streaming sensor data and (2) aggregating and streaming real-time data to components of a distributed system. We tackle these problems in the context of developing an emergency response system.

In this chapter, we discuss the problem of emergency response management and survey emergency response systems. We also present an overview of the Wireless Phone-based Emergency Response (WIPER) project and related research problems.

1.1 EMERGENCY RESPONSE MANAGEMENT

Emergency response managers decide how to use available resources to cope with a crisis situation. Making good decisions is difficult, especially when factors such as stress, fatigue, and restrictive time constraints are present. These stressors are common in emergency situations. Emergency response systems provide support for gathering, analyzing, and presenting data in such a way that it is useful and meaningful to the manager [51].

1.2 EMERGENCY RESPONSE SYSTEMS

Emergency response systems provide communication, data collection, data analysis, and decision making tools. The standard model for emergency response systems consists of four components that provide the following functionality: data storage with fast access, fast data analysis, solution evaluation, and a user interface that provides access to all other components [51].

Belardo *et al.* [20] describe four early emergency response systems.

- A system for the American Red Cross stores survey data collected before emergencies in high risk areas, initial damage survey results, and victim claim information. The goal of the system is to quickly verify claims to facilitate timely aid distribution.

- A system for the United States Coast Guard predicts the position of a lost vessel based on the last known position and drift forces such as current and wind. Based on the result, the Coast Guard implements a search pattern.

- A system for a regional emergency medical organization provides information on available resources, modeling tools to aid in resource allocation, and reminders of correct procedures.

- A system for the New York State Office of Disaster Preparedness, motivated by the Three Mile Island incident, includes tools for storing and accessing information on available resources, including equipment and personnel, and modeling the dispersion of radioactive material over time.

Current technology can provide emergency response managers access to abundant data; however, studies have shown that abundant data does not necessarily lead to good decisions. In cases where there is more data than can be effectively handled, a phenomenon referred to as *information overload*, ad hoc processes for reducing the data volume for human processing may distort the data causing a degradation in decision quality [84]. Emergency response systems attempt to counteract information overload by providing methods of organizing and presenting data such that more data can be effectively used by decision makers.

1.3 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 system, a sensor network, and a software system that provides data from the sensor network to the simulation system as requested [34].

Darema [34] 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 must have robust convergence properties under dynamic data inputs.
- Systems software must support dynamic computational, communication, and data with fault tolerance and quality of service guarantees.
- 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 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 [39].

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 [40].

1.4 OVERVIEW OF THE WIPER PROJECT

The Wireless Phone-Based Emergency Response (WIPER) system, currently under development, utilizes a cell-phone network as a set of sensors for gathering and presenting information to emergency response managers. The system monitors the network data for anomalous activity, runs simulations to predict population movement during a crisis, and provides the emergency response manager with a current view of the affected area using GIS tools [60, 61, 77, 78].

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 and detects anomalous activity. There are various aspects of the cell-phone network

data that may be of interest, include overall usage levels, spatial distribution of users, and social network characteristics.

- 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 cellular 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.

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

Figure 1.1 shows an architectural overview of the WIPER system. The RTDS and HIS provide the bridge from the service provider, about which we have very little architectural knowledge, 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 Detection and Alert System to inform the Simulation and Prediction system of a potential anomaly in the streaming data.

1.5 OVERVIEW OF PROPOSED RESEARCH

The research proposed in this document focuses on two components of the WIPER system: the Detection and Alert System and the Real-Time Data Source. Our goal is to

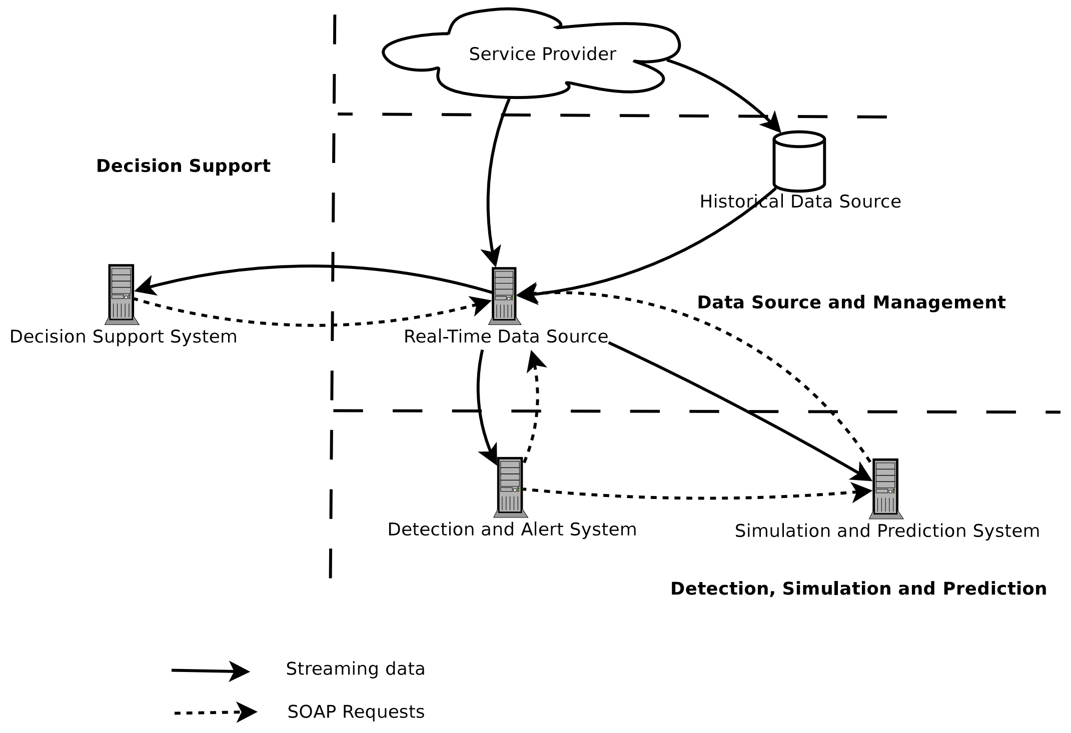


Figure 1.1. WIPER system architecture.

develop:

- fast, dynamic, stream mining algorithms for anomaly detection in both relational and non-relational data
- a real-time system that distributes data from the service provider to the WIPER components quickly and reliably.

CHAPTER 2

BACKGROUND

2.1 OVERVIEW

In this chapter, we present background information relevant to development of the Detection and Alert System and Real-Time Data Source for WIPER. We survey several topics, including outlier detection, anomaly detection in time series data, complex networks, link mining, data stream models, and real-time system.

2.2 OUTLIER DETECTION

An outlier is an item in a data set that does not appear to be consistent with the rest of the set [17]. Outliers are of interest because they may be noise that significantly degrades the quality of data models. There is a great deal of literature on the problem of outlier detection as well as a number of applications, including fraud detection, intrusion detection, and time series monitoring [46].

2.2.1 Methods for Finding Outliers in Data

There are three fundamental approaches for outlier detection [46]:

- *Model normality and abnormality.* This approach assumes that a training set representative of both normal and abnormal data exists.

- *Model one of normality or abnormality.* This approach typically models normality and is well suited for dynamic data.

- *Assume no a priori knowledge of the data.* This approach is well suited for static distributions and assumes that outliers are, in some sense, far from normal data.

There are several statistical methods for outlier detection, including parametric methods, semi-parametric methods, non-parametric methods, and proximity based methods [46].

2.2.1.1 Parametric Methods

Parametric outlier detection methods assume that the data follows a particular probability distribution. These methods tend to be fast but inflexible. They depend on a correct assumption of the underlying data distribution and are not suitable for dynamic data [46].

Statistical process control assumes that the variation in the output of a process follows a Gaussian distribution. In manufacturing, a certain amount of deviation in some measure on the output of a process is expected due to variations in material and other factors; however, a significant deviation from the mean may indicate that the equipment is not working properly either due to a mechanical failure or wear on the system over time. The boundary between random variation, the former, and assignable variation, the latter, is $\mu \pm k\sigma$, where μ is the mean, σ is the standard deviation, and k is some constant. [22].

2.2.1.2 Semi-parametric Methods

Semi-parametric models use mixture models or kernel density estimators rather than a single global model. Both mixture models and kernel models estimate a probability distribution as the combination of multiple probability distributions.

Mixture models consist of a weighted sum of probability density functions. Gaussian mixture models, which consists of a set of Gaussian probability density functions, are most common, though any combination of probability density functions may be used [87]. The expectation maximization clustering algorithm discovers a Gaussian mixture model for a data set where each Gaussian probability density function corresponds to a cluster.

A kernel density estimator models the data set by assuming a probability density function is centered at each point in the dataset. The probability density function for the entire data set is the sum of all the kernel estimators [79]. Yeung and Chow [95] use a kernel density estimation, with Gaussian kernels, as a basis for hypothesis testing for intrusion detection.

2.2.1.3 Non-parametric and Proximity Based Methods

Non-parametric methods make no assumptions about the underlying distribution of the data and tend to be computationally expensive. Many non-parametric methods define outliers in terms of their distance from other points in the data set. Specific definitions vary from application to application. Two definitions that appear in the literature are stated below.

Definition 1 (from [54])**.** An object O in a dataset T is a $\mathsf{DB}(p, D)$ outlier if at least fraction p of the objects in T lie greater than distance D from O

Definition 2 (from [75])**.** The D_n^k outliers are the n points with the largest distance from their kth nearest neighbor.

Both DB and D_n^k outliers can, naïvely, be discovered by computing the distances between all points in the data set and finding the points that meet the criteria specified in above definitions; however, this approach is slow, especially for large data sets. The detection of these outliers may be accelerated with the use of spatial data structures. Knorr and Ng [54] present a cell-based method that divides the space into hypercubes of equal size. The properties of this structure, with a carefully chosen cell size, allow all points in certain cells to be identified as outliers or non-outliers at once, in many cases. Multiresolutional kd-trees [66] may also be used to speed up the search for outliers. Mrkd-trees form a hierarchical representation of the points in a data set, *i.e.*the root of the tree "owns" all nodes and a node's points are partitioned among its two children. Each node stores basic statistics on the points it "owns": the number of points, the centroid, the covariance, and the bounding hyper-rectangle. Chaudhary *et al.*[27] present a method of outlier detection that computes a mr$k$d-tree such that the points in each node are divided evenly among its children and refines the node boundaries so that each node has a uniform density distribution of points. This produces a set of nodes in which all points in a node have the same amount of "outlier-ness".

2.2.2 Methods for Finding Anomalies in Time Series Data

Keogh *et al.* [52] discuss the problem of finding surprising patterns in time series data. They provide the following formal definition of a surprising pattern:

Definition 3 (from [52])**.** A time series pattern P, extracted from database X is surprising relative to a database R if the frequency of its occurrence is greatly different to that expected by chance, assuming that T and X are generated by the same underlying process.

The approach discretizes the time series into a symbolic string (the specific discretization method depends on the application) and uses either a Markov model or a suffix tree to characterize the probability of each substring.

Lin *et al.* [58] describe a discretization and dimensionality reduction method for time series data called SAX. They also define a measure that lower-bounds the distance between two time series that have been discretized via SAX. This result makes SAX feasible for use with various machine learning techniques such as clustering, nearest neighbor classification, and decision tree classification.

Wei *et al.* [91] use SAX in conjunction with a pair of sliding windows for assumption-free anomaly detection. They use a "lag" window as a base line and compute its distance from the "lead" window. This distance provides a measure of "anomalousness" for the "lead" window at each time step.

Scott [80] presents a method for network intrusion detection using a Markov modulated nonhomogeneous Poisson process. The approach consists of two Poisson processes, one modeling normal behavior and another modeling abnormal behavior, and a Markov process modeling the probability of moving from normal behavior to abnormal behavior and vice versa. Scott and Symth [81] use a similar method for modeling web traffic, and [47] use a Markov modulated Poisson process for modeling foot traffic at the entrance of a building and freeway traffic near a major sport venue.

2.2.3 Application: Fraud Detection

Outlier detection is often used in fraud detection applications. Identifying fraud quickly is important in order to minimize the loss due to the fraud. Fraud is pervasive and takes many forms, including money laundering, credit card fraud, telecommunications fraud, computer intrusion, medical fraud, and scientific fraud [24].

We discuss two approaches to fraud detection. The supervised approach requires data describing non-fraudulent behavior and all classes of fraudulent behavior that are to be detected. The unsupervised approach identifies outliers as potential fraud and flags these for further investigation [24].

Fawcett and Provost [36] present a supervised framework for detecting cloning fraud, a form of telecommunication fraud in which identifying account information is stolen and used to gain access to a legitimate user's account. Their approach uses a rule based system to identify fraudulent calls and a set of profile monitors that model normal behavior and variation for each user.

2.2.3.1 Intrusion Detection

Intrusion detection is a form of fraud detection which seeks to quickly identify any security compromise of a computer system in order to minimize the damage inflicted by a malicious user. A security compromise occurs when an individual obtains permissions on a machine to which he is not entitled. There are many ways in which a compromise can occur, including a stolen password and exploitation of vulnerable programs. There are two major approaches to intrusion detection: signature detection and anomaly detection [13].

The signature detection approach requires each type of intrusion to be specified. The major drawback of this method is that it can only detect known intrusions and must be updated as new types of intrusions are discovered. The anomaly detection method for intrusion detection tend to be semi-supervised and are based on the assumption that "abnormal behavior is probably suspicious" [13].

Denning [35] suggests the idea of viewing abnormal behavior as suspicious based on the observation that many security compromises have been discovered by system administrators and users who observe strange behavior. Various approaches of anomaly detection have been applied to this problem.

Some approaches utilize standard methods for detecting anomalies, such as hidden

Markov models [96] and clustering [74]. Other approaches are based on hypothesis testing and entropy. Yeung and Ding [96] use cross entropy as a measure of the difference between the distributions of two data sets as the basis for hypothesis testing.

2.3 COMPLEX NETWORKS

A wide range of phenomena may be modeled as graphs, *e.g.* social interaction, biological systems, the world wide web, research collaboration, power grids, transportation systems, food webs, and ecosystems. [9, 15, 23]. In this section we introduce standard graph metrics and major graph models. We also discuss approaches for data mining complex networks.

2.3.1 Graph Metrics

2.3.1.1 Diameter

The *diameter* of a graph is the longest shortest path in a graph and gives the maximum distance required to travel between any two vertices in the graph. The diameter of an unconnected graph is ∞; however, the maximum component diameter may be used in this situation [94].

2.3.1.2 Degree Distribution

One of the most commonly measured graph characteristics is the *degree distribution*, which is the probability distribution of the degrees of the vertices. Several network models produce graphs with a Poisson degree distribution, including the Erdös-Rényi and Watts-Strogatz models discussed below, and many real world networks are known to have power-law degree distributions [9]. Under a power-law distribution, the probability a vertex has degree k is

$$P(k) \propto k^{-\gamma} \tag{2.1}$$

where $\gamma > 1$ [16].

2.3.1.3 Characteristic Path Length

The *characteristic path length* of a graph is the average shortest path length between a pair of vertices [90]. A graph is characterized as a *small-world* network if its characteristic path length is no greater than logarithmic with respect to the number of vertices in the graph [23].

2.3.1.4 Clustering Coefficient

The *clustering coefficient* characterizes the connectivity of the neighborhoods of a graph. Two methods of computing the clustering coefficient appear in the literature.

The first method, by Watts and Strogatz [90], is the average probability that two neighbors of a vertex are connected. Let e_v be the number of edges connecting neighbors of vertex v and $d(v)$ be the number of neighbors for vertex v (the degree of v). The clustering coefficient is

$$C = \frac{1}{|V|} \sum_{v \in V} \frac{e_v}{d(v)\left(d(v) - 1\right)/2} \tag{2.2}$$

The second method, by Newmann [68], computes the probability that two neighbors of any vertex are connected.

$$C = \frac{\sum_{v \in V} e_v}{\sum_{v \in V} d(v)\left(d(v) - 1\right)/2} \tag{2.3}$$

Readers should be aware that both methods are used in the literature and may produce different values. The definition by Watts and Strogatz gives greater weight to vertices with smaller degrees [70].

2.3.2 Graph Models

2.3.2.1 Erdös-Rényi Model

Erdös and Rényi study the properties of evolving random graphs of $|V|$ vertices as the probability, p, of a an edge existing between any pair of vertices increases. They find that a number of properties appear suddenly, with high probability, at some critical probability, p_c, as $p \rightarrow 1$. One of their most famous discoveries is the critical probability for the formation of a giant component, *i.e.* a component of size $O(|V|)$, is $1/|V|$. When $p > 1/|V|$ a component with $O(|V|)$ vertices is likely present. When $p < 1/|V|$, a random graph likely has $O|E|$ components, none of which have more than $O(\ln |V|)$ vertices. When $p = 1/|V|$, the largest component of the graph is probably $O(|V|^{2/3})$ vertices. Erdös-Rényi graphs typically have small characteristic path length and small clustering coefficients [9, 23, 70].

2.3.2.2 Watts-Strogatz Model

Watts and Strogatz [90] observe that naturally occurring networks are neither completely random, like the Erdös-Rényi model, nor completely ordered. The Watts-Strogatz generative model starts with a regular lattice. The vertices are organized in a circle and each vertex in the graph is connected to some constant number of its nearest neighbors, resulting in a regular graph in which each vertex has the same degree. Each edge in the lattice is randomly rewired with a constant probability. Watts and Strogatz evaluate their model using two metrics: characteristic path length and clustering coefficient. They show that while both random graphs and graphs generated using their model both exhibit the small-world characteristic, their model produces graphs with the larger clustering coefficients that are found in naturally occurring graphs.

2.3.2.3 Barabási-Albert Scale-Free Graphs

Albert, Jeong, and Barabási [10] show that, like social and biological networks, the world wide web is a small world network and has an estimated characteristic path length of 19. Additionally, they discovered that the world wide web has a power law distribution, a distribution that is significantly different than the expected Poisson distribution. Further investigation reveals power law distributions in actor collaboration and power grid graphs. Graphs with a power law degree distribution are referred to as *scale free* graphs because their properties are independent of the number of vertices [10, 15, 16].

2.3.3 Link Mining

Link mining learns on a collection of related objects rather than assuming each item in a data set is independent—an assumption of traditional data mining methods. Link mining tasks can be divided into three types [41]:

- *Object related tasks* include link based ranking, link based classification, group detection, and object identification.
- *Link related tasks* include link prediction and anomalous link detection.
- *Graph related tasks* include subgraph discovery, graph classification, and generative models for graphs

Generative models for graphs include the Erdös-Renyi, Watts-Strogatz, and Barabási-Albert models discussed above. The link prediction and anomalous link detection problems are directly related to the research presented in this proposal, and we briefly discuss them below.

2.3.3.1 Link Prediction Problem

Liben-Newell and Klienberg [57] define the link prediction problem:

> Given a snapshot of a social network, can we infer which new interactions among its members are likely to occur in the near future?

They evaluate various approaches for assigning a connection weight rank for each pair of nodes in the graph. The ranks may be a function of proximity or similarity and are determined by examining either the neighborhoods within the network or the paths through the network. Neighborhood based ranks may be the number of neighbors shared by two nodes, the probability that two nodes share a given neighbor (Jaccard's coefficient), or the product of the number of neighbors of two nodes (preferential attachment). Path based ranks may be a weighted sum of the number of paths between two nodes such that shorter paths are assigned higher weights (Katz method) or the expected number of links between two nodes (hitting time) [57].

2.3.3.2 Anomalous Link Detection

The anomalous link detection problem consists of identifying links that are not likely to occur. Rattigan and Jensen [76] applied the Katz method, used for link prediction by Liben-Nowell and Kleinberg [57], to detect anomalous links.

2.4 DATA STREAMS

Data stream model deals with data sets that may only be read once and only in the order in which they are generated. These limitations are due to the volume of the data; the data sets are so large that it is not feasible store the entire data set in main memory, read the data multiple times, or perform random accesses on the data. A data stream may be the product of reading a large file from disk or tape, the result of a database query, or the output of an ongoing process. Given size of data streams and the memory restrictions, streaming algorithms are required to use sublinear space. In general, algorithms using polylogarithmic space with respect to the number of items in the data stream are considered good solutions [14, 45].

2.4.1 Measuring the Evolution of Streaming Data

A characteristic of a data stream we may wish to consider is the way in which it evolves over time. Arggarwal [5] suggests two metrics for measuring the evolution of a stream: (1) velocity density estimation, which estimates the rate at which the density of data at a given location is changing and (2) spatial vector profiles which provides insight into the manner in which the data is shifting. Kifer *et al.*[53] use a “two window paradigm” in which a *reference window* is used as a baseline with which future data is compared and the *current window*. This method uses a generalization of the Kolmogorov-Smirnov statistic to estimate the distance between the distributions that produced the values in the two windows.

2.5 REAL-TIME SYSTEMS

Real-time systems behave according to explicit timing constraints and are generally divided into two categories.

- *Hard real-time systems* fail if any time constraint is violated.
- *Soft real-time systems* tolerate some number or rate of time constraint violations.

A real-time system consists of a set of *tasks*, each of which is a process or thread. Each task consists of a sequence of *jobs*. There are three types of tasks.

- *Periodic tasks* release jobs at regular intervals.
- *Aperiodic tasks* release jobs at irregular intervals.
- *Sporadic tasks* are aperiodic tasks in which the release of two consecutive jobs is constrained by a minimum time interval.

The tasks in a real-time system are assigned a priority, which determines the order in which tasks are executed. Tasks with higher priorities run before tasks with lower priorities.

2.5.1 The Periodic Task Model

Liu and Layland [59] define the canonical real-time model for periodic tasks. The model is highly constrained; they assume that all tasks are periodic with a deadline equal to the period, all tasks are independent, all tasks have a fixed or bounded computation time, all tasks may be preempted so that a task with a higher priority may run, and all tasks run on a single processor [59, 82].

In the periodic task model, the jobs of a task are defined by four parameters.

- The *release time* is the earliest time a job may begin execution
- The *execution time*, e, is worst case CPU time required to execute the job.
- The *deadline*, d, is the time by which the job execution must be completed.
- The *period*, p, is the rate at which the task releases jobs. The deadline of a job is equal to the period of its task.

In their paper Liu and Layland [59] define two fundamental scheduling algorithms and derive feasibility conditions, the conditions under which a task set meets all temporal requirements. Their analyses are based on the concept of a *critical instant* which is a time at which a job will require maximum time to complete. A critical instant occurs when a job is released at the same time that all higher priority tasks release a job. If all jobs meet their timing constraints at their critical instant, the task set is feasible.

The feasibility analyses are based on the *system utilization*, fraction of available processing time required by a task. For task i with execution time e_i and period p_i, the utilization is e_i/p_i. The total utilization, U, of a system with m tasks is

$$U = \sum_{i=1}^{m} \frac{e_i}{p_i}.$$

Rate monotonic scheduling is a fixed priority algorithm in which the priority of a task is inversely related to its period. Liu and Layland [59] derive a sufficient feasibility condition for rate monotonic scheduling; a set of m periodic tasks is feasible if

$$U \leq m\left(2^{1/m} - 1\right). \tag{2.4}$$

Earliest deadline first scheduling is a dynamic priority scheduling algorithm. in which the priority of a task is inversely related to the time until its next deadline. Liu and Layland [59] show that with earliest deadline first scheduling, there can be no idle processor time, *i.e.* the processor is fully utilized, prior to any missed deadline. They, therefore, derive a necessary and sufficient feasibility condition for earliest deadline first scheduling; a set of m periodic tasks is feasible if and only if

$$\sum_{i=1}^{m} \frac{e_i}{p_i} \leq 1. \tag{2.5}$$

2.5.1.1 Extensions of the Liu and Layland Model

The work of Liu and Layland [59] provides an important foundation in real-time scheduling research, and much work has been done to extend this model.

Deadline monotonic scheduling is a generalization the rate monotonic scheduling algorithm in which the deadline of a periodic task is less than or equal to its period. This generalization improves the schedulability analysis for task sets where the worst case execution times are less than the periods [12].

Several methods have been developed for improving the response time of aperiodic tasks, which do not typically have hard deadlines but may have quality of service, average response time, or average throughput requirements. Several solutions utilize a polling server, a periodic task that provides time in which aperiodic tasks execute. The more sophisticated server methods are able to use more processor time when the total system utilization is low [82].

2.5.2 The Sporadic Task Model

The sporadic task model [65] assumes that all tasks are sporadic. A sufficient feasibility test for a sporadic task system assumes that each task is released as frequently as allowed by the system, in which case the analysis degenerates into the periodic case; however, this approach does not permit a large number of feasible task sets. Baruah *et al.* [19] derive a necessary and sufficient feasibility test for sporadic task sets. While their algorithm for determining feasibility takes exponential time in the worst cases, it requires pseudo-polynomial time in most cases. Baruah and Fisher [18] derive a polynomial time feasibility test for sporadic real-time task sets on single and multiple processor systems when there are a constant number of distinct task types.

2.5.3 Rate-Based Execution

Jeffay and Goddard [50] note that in practice, distributed real-time systems are often neither periodic nor sporadic and that in many cases the expected rate at which jobs are released in known. To handle this situation, they develop a model for rate-based execution.

Tasks in the rate-based execution model are defined by four parameters:

- a time interval, y
- the maximum expected number of times the task releases jobs, x, in a time interval y
- the maximum desired response time, d, of the jobs
- the maximum execution time, e.

The deadline of each job depends on either the start time or the deadline of the xth prior job. When jobs are released at a rate less than the maximum expected rate, the deadline of a job is the sum of its release time and its desired response time. When jobs are released at a rate greater than the maximum expected rate, only the first x jobs in a time interval of length y are expected to complete within d time, the remaining tasks are allowed y time beyond the deadline of the xth previous task. Formally, let t_{ij} be the release time of the jth job or the ith task. The deadline for job j of task i is

$$D_i(j) = \begin{cases} t_{ij} + d_i & \text{if } 1 \leq j \leq x_i \\ \max\left(t_{ij} + d_i, D_i\left(j - x_i\right) + y_i\right) & \text{if } j > x_i \end{cases} \tag{2.6}$$

Jeffay and Goddard [50] derive necessary and sufficient feasibility conditions for rate based execution under preemptive scheduling, with and without shared resources, and non-preemptive scheduling.

CHAPTER 3

ANOMALY DETECTION IN A MOBILE COMMUNICATION NETWORK

3.1 ABSTRACT

Mobile communication networks produce massive amounts of data which may be useful in identifying the location of an emergency situation and the area it affects. We propose a one pass clustering algorithm for quickly identifying anomalous data points. We evaluate this algorithm's ability to detect outliers in a data set and describe how such an algorithm may be used as a component of an emergency response management system. [1] [2]

3.2 INTRODUCTION

Cellular networks have recently received attention as viable, pre-existing sensor networks. City officials in Baltimore use cell phone location data to monitor traffic flow, and the state of Missouri is considering a similar state wide program that would make traffic information available to the public [11]. IntelliOne, a company based in Atlanta, GA, recently released a system that displays anonymous cell phone location data onto a

[1]This paper won the best student paper award at the North American Association for Computational Social and Organizational Science (NAACSOS) Conference 2006, University of Notre Dame, Notre Dame, Indiana, USA. [72]

[2]This work has been accepted for publication in Computational & Mathematical Organization Theory. [71]

map so that users can identify congested areas [67]. The emergency response community has also gained interest in using existing cell phone networks as a way to distribute warnings to citizens [83, 93].

The Wireless Phone Emergency Response (WIPER) system, an emergency response management tool currently under development, monitors an existing cellular phone network. Operating under the assumptions that the behavior of the network models the behavior of a population and that anomalous behavior may indicate an emergency situation has developed, the system attempts to quickly detect anomalies in the network. When anomalies occur, WIPER uses a suite of simulations to predict how the situation will unfold. This paper focuses on the problem of identifying anomalous events in streaming cell phone data as part of the WIPER system. See [61, 77, 78] for a complete overview of the WIPER system.

Our goal is to mine the cellular phone network data for events and anomalies to enable a more efficient emergency response system. Emergency response systems are tools that aid emergency response managers in the decision making process. The difficulty of making good decisions is increased by several factors including stress, fatigue, restrictive time constraints.

Another major issue for emergency response managers is the problem of "information overload". Studies have shown a correlation between abundant available data and bad decision making in crisis situations [84]. Good emergency response systems provide access to the large amount of available data in such a way that the emergency response manager can use the data effectively to reach good decisions [20, 51]

We believe that an anomaly detection system that monitors incoming streaming cellular network data and posts alerts for the emergency response manager will be a useful addition to an emergency response system. It will draw the managers attention to information that may be easily missed in a fast moving, stressful situation. The manager can use or ignore that information based on their experience. The goal is to provide valuable information without being too intrusive in the case of false positives.

The nature of the data poses some difficulties in developing an anomaly detection system. First, a large amount of data arrives at a rapid rate. The sheer volume of the data makes it difficult to store it in its entirety, much less operate on it using repeated accesses, which is a typical algorithmic requirement. Therefore, we aim to develop a method that follows the data stream model [14]. Intuitively, a data stream is a sequence of data items that arrive at such a rapid rate that it is only feasible to operate on a small portion of the data. As each data item is seen, it must be either incorporated into a summary that requires a small amount of memory or it must be discarded, in which case it cannot be retrieved. The data stream model imposes two algorithmic limitations: each item in the dataset may only be read once in a predefined order, and memory usage must be sub-linear—typically polylogarithmic with respect to the number of data items seen. Our main focus in this paper is the one pass requirement; we present a one pass hybrid clustering algorithm for anomaly detection.

Another difficulty is the fact that the system is dynamic; the way in which people use the services provided by a cellular network changes over time. The anomaly detection approach should be sensitive enough to detect anomalies but should not be so sensitive that it flags changes in the underlying system as anomalies. That said, the cost of false positives is far less than the cost of false negatives. The system can handle the detection of a few non-emergency situations, as long it does not happen too often.

In this paper, we present a one-pass hybrid clustering algorithm for detecting anomalies in streaming data. We evaluate clusters produced by the algorithm and its ability to detect outliers. Finally, we discuss how such an algorithm can be used in an emergency response system like the one described above.

3.3 RELATED WORK

There is abundant literature on the anomaly detection problem which describes a variety approaches, including statistical, neural network, and machine learning methods. In this paper, we focus on the statistical approaches, which can be divided into two categories: parametric and non-parametric. Parametric approaches tend to be more efficient, but assume the data conforms to a particular distribution. Non-parametric methods do not assume any particular data distribution; however, they are often less of efficient. [46, 62, 63].

Clustering is an appealing non-parametric method because it allows us to capture various classes of “normal” and “abnormal” behavior. This may be quite useful since, in addition to detecting anomalies caused by events that have never been seen before, knowing various types of “abnormality” would allow us to identify interesting events that have already been seen.

The goal of clustering is to group similar data items together. The concept of similarity is often defined by a distance metric; we use Euclidean distance. Good clustering algorithms form clusters such that the distance between intra-cluster points are minimized and the distance between inter-cluster points are maximized. Anomalies are, intuitively, the data items that are far from all other data items. There are three major types of clustering algorithms: partitional, hierarchical, and incremental [48].

3.3.1 Partitional Clustering

Partitional clustering divides the data set into some number, often a predefined number, of disjoint subsets. K-means is a classical and simple clustering algorithm that iteratively refines a set of clusters. The initial cluster centroids for the k-means algorithm are k randomly selected data items from the data set. Each example in the data set is assigned to the closest cluster, and the new cluster centroids are computed. This process is repeated until the clusters stabilize, *i.e.*no point in the data set receives a new cluster assignment [92].

Expectation maximization (EM) clustering is another classical, partitional algorithm. EM is a probability based algorithm that seeks to discover a set of clusters corresponding to a Gaussian mixture model, a set of Gaussian distributions, that describes the data set. The algorithm is initialized with k random Gaussian distributions and iteratively refines these distributions using a two step process. The expectation step computes the probability that the data set is drawn from the current Gaussian mixture—the likelihood. The maximization step reassigns the data items to the cluster which they most likely belong and recomputes the Gaussian mixture. The algorithm halts when the likelihood that the dataset is drawn from the Gaussian mixture increases by less than a user defined threshold.

There are a couple of drawbacks with these approaches. The k-means and EM algorithms are not guaranteed to find an optimal set of clusters, and both algorithms require *a priori* knowledge of the number of clusters in the data. These issues can be mitigated by running the algorithms multiple times using different initial conditions and varying numbers of clusters. The best set of clusters is used to describe the data [92]. Another issue is scalability. These algorithms are inefficient for very large data sets. Spatial data structures may reduce the time required by these algorithms. kd-trees [21] have been used reduce the number of distance calculations required by k-means. Often, a kd-tree can be used to determine cluster memberships for a subset of points with only k distance computations (rather than k computations for each point in the subset) [73]. Multiresolutional kd-trees have been used to improve the performance of EM clustering. A multiresolutional kd-tree stores hierarchical summary statistics on the data “owned” by the node: the number of points, centroid, covariance matrix, and bounding hyperrectangle. With these summaries stored for hierarchical subsets of the data set, the computation of EM parameters can be accelerated significantly [66].

3.3.2 Hierarchical Clustering

Hierarchical clustering divides data into a nested set of partitions and may be useful for discovering taxonomies in data. Agglomerative algorithms produce hierarchical clusters via a bottom-up approach in which each example is initially a unique cluster and the clusters are iteratively merged with their neighbors. Two common agglomerative clustering algorithms are single link and complete link. These algorithms are graph based: each example becomes a vertex, and edges are added based on the distance between pairs of vertices. A level of the hierarchical cluster is defined by a distance threshold: an edge is added to the graph if and only if two examples are separated by a distance less than the threshold. The connected components and completely connected components are the clusters for the single link and complete link algorithms, respectively. The hierarchy of clusters is formed by iteratively increasing the threshold to produce larger clusters [48, 49]. Since single and complete link clustering compute the distances between all pairs of examples in the dataset they have greater time complexity than partitional algorithms, however, they produce optimal solutions.

3.3.3 Incremental Clustering

Incremental algorithms consider each example once, immediately deciding either to place it in a existing cluster or to create a new cluster. These algorithms tend to be fast, but are also often order dependent [48]. 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 [44].

Portnoy *et al.* use the leader algorithm for intrusion detection (another application of anomaly detection. In order to handle arbitrary distributions, they normalize the data using z-score, in which the feature values are transformed by

$$v_i' = \frac{v_i - \bar{v}_i}{\sigma_i} \tag{3.1}$$

Unfortunately, this requires two passes over the data. Furthermore, the distance threshold is fixed over all clusters, and cannot change as the data stream evolves.

3.3.4 Clustering Algorithms for Streaming Data

A few methods have been developed for clustering data streams. Guha *et al.*[43] 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.*[7] 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 computes *micro-clusters*, which are statistical summaries of the data periodically throughout the stream. These micro-clusters serve as the data points for a modified k-means clustering algorithm.

3.3.5 Hybrid Clustering

Hybrid clustering combines two clustering algorithms. Cheu *et al.*[28] 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 [29] 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.*[86] propose a hybrid clustering algorithm for document classification that uses hierarchical clustering as a method for determining initial parameters for expectation maximization.

3.4 THE DATASET

We use a data set generated from a database of real world cellular network information. The database provides the following information for each transaction (use of a service by a customer): the initiation time, the duration (in minutes), and the name of the service. From the database, we generated a data set with 17,280 examples. Each example indicates how many instances of each service are in use for each minute of a 12 day period. We prune the year, month, and day from the data set due to the small time frame covered and we remove 11 services that are rarely used. This leaves a data set with 7 features: hour, minute, data transmission usage, general packet radio service (GPRS) usage, telephone usage, text messages sent, and text messages received.

Figure 3.1 shows the time series for each of the seven service features of the dataset. Note that each time series exhibits periodic concept drift, an underlying change in the process generating the data [88], based on the time of day. The telephone time series is relatively consistent from day to day, though the call volume varies somewhat depending on the day of the week and on whether the day is a holiday. In contrast, there is a noticeable increase in the network load for each of the other services as time goes by; this is a form of non-periodic concept drift. This suggests that the way in which people use the telephone service is relatively well established. Notably, this is also the oldest and most used service. As technology evolves, and peoples habits change, we can expect new manifestations of concept drift.

3.4.1 Offline Clustering Analysis

Since many clustering algorithms require *a priori* knowledge of the number of clusters, we must have some way of determining the correct value for this parameter. There are a couple of methods for accomplishing this. One method is to simply perform the clustering for various numbers of clusters and choose the best result based on some metric such as sum squared error or log likelihood. Another method is to use 10-fold cross validation for $k \in 1, 2, \ldots, m$, increasing k until the quality of the clustering starts to degrade [92].

We use the implementation of expectation maximization provided by the Weka package [92] with 10-fold cross validation to determine the number of clusters. 10-fold cross validation partitions the data set into 10 equally sized subsets, or folds. Starting with $k = 1$, for each distinct set of 9 folds we compute clusters and the log likelihood of the cluster set. The value of k is incremented by 1 and the process repeats until the average log likelihood is less than that of the previous iteration. The final result is the set of clusters that maximizes the average log likelihood. While this approach is not necessarily likely to find a global maxima, it is consistent with Occam's Razor in favoring a smaller number of clusters, which corresponds to a simpler hypothesis [92].

We use expectation maximization to cluster the dataset in the following two ways.

First, we arbitrarily select a day from the data set (day 4) and compute the clusters for each hour of the day (see figure 3.2). Second, we compute clusters for accumulated data. We cluster the first day, the first two days, the first three days, and so on until we include all 12 days of data (see figure 3.3). Using both approaches, we find that the number of clusters fluctuates, indicating that the appropriate value for k changes as the stream progresses.

3.5 A HYBRID CLUSTERING ALGORITHM

We implement a hybrid clustering algorithm that combines a modification of the leader algorithm with k-means clustering. The basic idea behind the algorithm is to use k-means to establish a set of clusters and to use the leader algorithm in conjunction with statistical process control to update the clusters as new data arrives.

Statistical process control [22] 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, μ, of that measure. The threshold is typically some multiple, l, of the standard deviation, σ. Therefore, if the measured output falls in the range $\mu \pm l\sigma$, the variance is considered random; otherwise, it is assignable.

Our algorithm represents the data using two structures: the cluster set and the outlier set. To save space, the cluster set does not store the examples that make up each cluster. Instead, each cluster is summarized by the the sum and sum squared values of its feature vectors along with the number of items in the cluster. The outlier set consists of the examples that do not belong to any cluster. We rely on the centroid and the standard deviations of the features to summarize and update the clusters, so clusters are only accepted when they contain some minimum number of examples, m. The algorithm periodically clusters the examples in the outlier set using k-means. Clusters which contain at least m items are reduced to the summary described above and added to the cluster set.

Algorithm 1 shows the details of our approach. The algorithm takes three arguments: the minimum number of elements per cluster, m, number of clusters to compute with k-means, k', and a threshold, l that, when multiplied by the magnitude of the standard deviation vector, defines the boundary between "random" and "assignable" variation. (Note that k' specifies the number of clusters for the first level of the hybrid algorithm, not the final number of clusters produced by the algorithm.) For each example that arrives, we compute the closest cluster. If the example is considered an "assignable" variation, *i.e.*it is further than $l\sigma$ from the closest cluster center (or the set of clusters is empty), the example is placed in the outlier set. Otherwise, if the example is considered a "random" variation, the example is used to update the summary of the closest cluster. When there are $k'm$ examples in the outlier set, cluster these examples with k-means. The new clusters with at least m examples are added to the cluster set, and all the examples in the remaining clusters return to the outlier set.

This algorithm attempts to take advantage of the fact that the mean is sensitive to outliers. By using means as the components of the cluster center and updating the centers whenever a new example is added to a cluster, we hope to handle a certain amount of concept drift. At the same time, we hope that the use of statistical process control to filter out anomalous data prevents the cluster centers from being affected by outlying points. This algorithm does not require *a priori* knowledge of the number of clusters (recall that the argument k' is only the number of clusters for the first level cluster), since new clusters will form as necessary.

Algorithm 1 INCREMENTALHYBRID(X, l, k, m)

```
Let $X$ be a list of examples, $\vec{x}_1, \vec{x}_2, \ldots$
Let $l$ be the threshold multiple
Let $k$ be the number of clusters to produce in the first level
Let $m$ be the minimum number of items required to accept a cluster

Let $C$ be the set of clusters
Let $U$ be the set of unclustered examples

$C \leftarrow \emptyset$
$U \leftarrow \emptyset$
for all $\vec{x} \in X$ do
    Find the closest cluster, $C_i$
    if $\mathsf{dist}(\vec{x}, C_i) < l|\vec{\sigma}|$ then
        Add $\vec{c}$ to $C_i$
    end if
    if $|U| = km$ then
        $C' \leftarrow k$-MEANS$(k, U)$
        for all $c' \in C'$ do
            if $c'$ contains more than $m$ examples then
                Add $c'$ to $C$
            else
                Put the items in $c'$ into $U$
            end if
        end for
    end if
end for
```

3.6 EXPERIMENTAL SETUP

We evaluate our incremental hybrid clustering algorithms against the expectation maximization clustering algorithm. We use the implementation of expectation maximization provided by the Weka package [92] using 10 fold cross validation to determine the baseline for the number of clusters in the data. For the hybrid algorithm, we use $l = 1, 3$ and $k' = 5, 10, 20, 30$. We evaluate the cluster quality using sum square error. We examine the number of clusters and outliers produced by the hybrid algorithm, and we compare the outlier set produced by the hybrid algorithm to outliers determined by an offline algorithm.

3.7 RESULTS

Figure 3.4 shows the number of clusters produced by expectation maximization clustering and our hybrid clustering algorithm. As expected, the number of clusters produced by the hybrid clustering algorithm decreases as the threshold, l, increases. Figure 3.5 shows the average number of outliers resulting from the application of the hybrid clustering algorithm, with error bars. There are a few factors that cause the number of outliers to fluctuate. Recall that we only accept clusters from k-means if they have some number of minimum members, m. Since we cluster when there are mk' members in the outlier set, increasing k' also increases the number of items used by k-means. If all the clusters are approximately the same size, several clusters with nearly m items may remain in the outlier set, increasing the number of outliers found by the algorithm. In contrast, if most of the examples fall in a few clusters, few examples may remain in the outliers set.

Figure 3.6 shows the sum squared error for expectation maximization and the hybrid algorithm. The hybrid algorithm produces clusters with less sum squared error, by orders of magnitude, than the expectation maximization algorithm. Also note that the sum squared error increases as both parameters, l and k' increase.

Figure 3.7 and 3.8 show the distribution of distances between points in the outlier set and their nearest neighbor in the full data set. Each figure also shows the nearest neighbor distance distribution for the full data set. Recall that we defined outliers as points in the dataset that are far from all other points. We define the extent to which a point is an outlier by its distance from its nearest neighbor. Data points that are closer to their nearest neighbor are less outlying that data points that are far from their nearest neighbor. Ideally, we would like the clustering algorithm to detect all extreme outliers in the nearest neighbor distance distribution for the full data set. These box plots show that this is not the case. The two most outlying examples are never found by the hybrid algorithm, and points below the first quartile in "outlier-ness" are regularly found. However, for some trials (specifically when $l = 3$ and $k' = 20, 30$), most of the outliers detected by the hybrid algorithm are extreme outliers (see figures 3.8c and 3.8d).

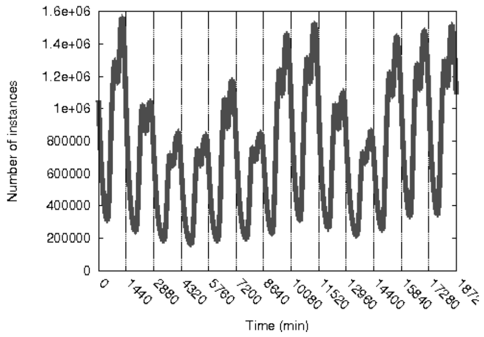
3.8 CONCLUSION

We have discussed issues in anomaly detection on dynamic data stream. We presented a hybrid clustering algorithm that combines k-means clustering, the leader algorithm, and statistical process control. Our results indicate that the quality of the clusters produced by our method are orders of magnitude better than those produced by the expectation maximization algorithm, using sum squared error as an evaluation metric. We also compared the outlier set discovered by our algorithm with the outliers discovered using one nearest neighbor. While our clustering algorithm produced a number of significant false positive and false negatives, most of the outlier detected by our hybrid algorithm (with proper parameter settings) were in fact outliers. We believe that our approach has promise for clustering and outlier detection on streaming data.

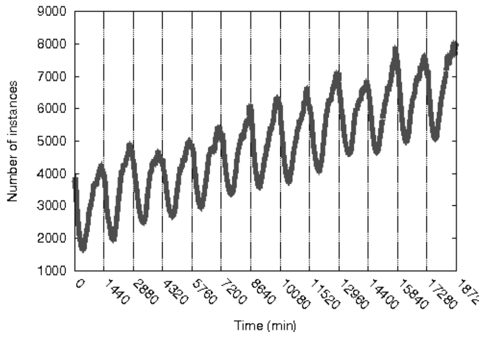
We also believe that this approach has promise for use as a component of the WIPER system. Determining where are new example will be placed—either in an existing cluster or in the outlier set—can be accomplished quickly. It is simply a matter of finding the closest cluster and determining if the example falls within its threshold boundary. Once an initial set of clusters is formed, an alert can be produced whenever a data item is assigned to the outlier set. Additionally, data items assigned to a cluster which is known to contain items produced during an emergency situation can also be used to inform the emergency response manager of a potential emergency.

3.9 PROPOSED RESEARCH

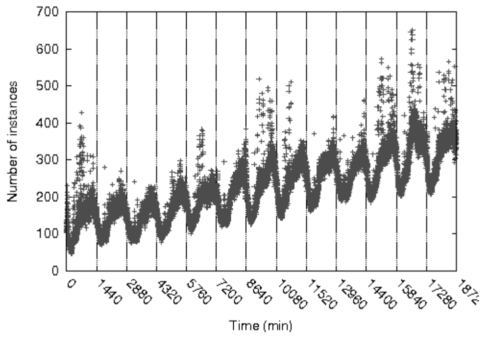
Further empirical tests have shown that a bad choice of parameters may cause an increase in the number false positives. We propose to explore the use of alternatives to k-means, particularly divisive algorithms which seem most intuitive for this application. We also propose to measure the changes in the clusters over time and develop methods for discarding old clusters without destroying the representation of long term patterns. Finally, in this paper we present an online clustering algorithm; we propose to further develop this algorithm so that it meets the space requirements for streaming algorithms.



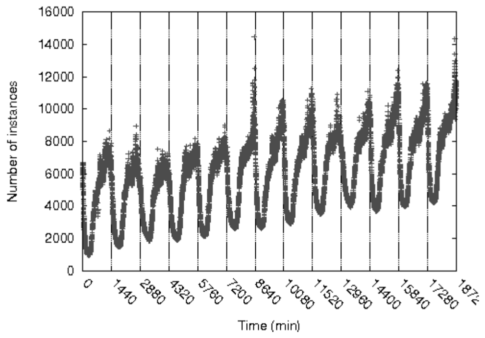
(a) Telephone



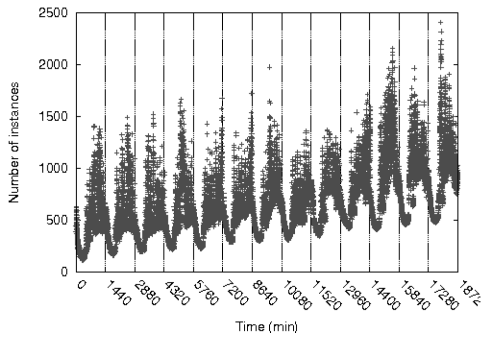
(b) Data Transmission



(c) GPRS



(d) SMS Sent



(e) SMSReceived

Figure 3.1. Time series for the five service features. These graphs show the number of times each service is used during each minute of the 12 day period.

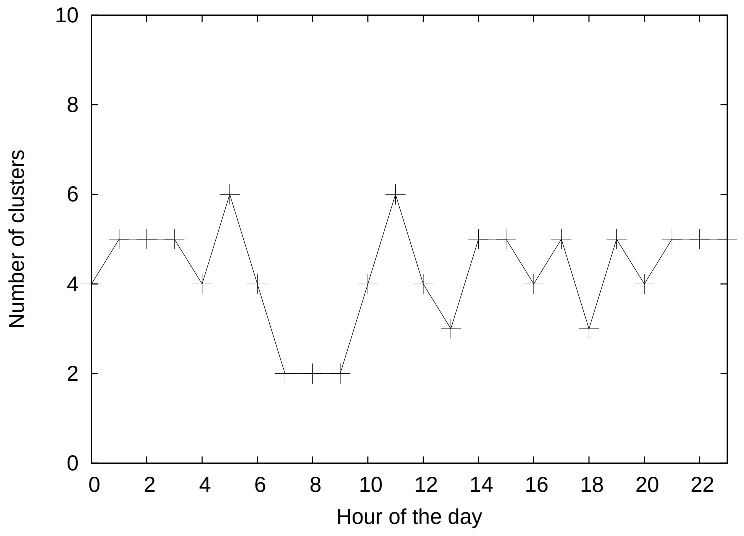


Figure 3.2. The number of clusters of for each hour of day 4. The number of clusters is determined using 10 fold cross validation with expectation maximization clustering.

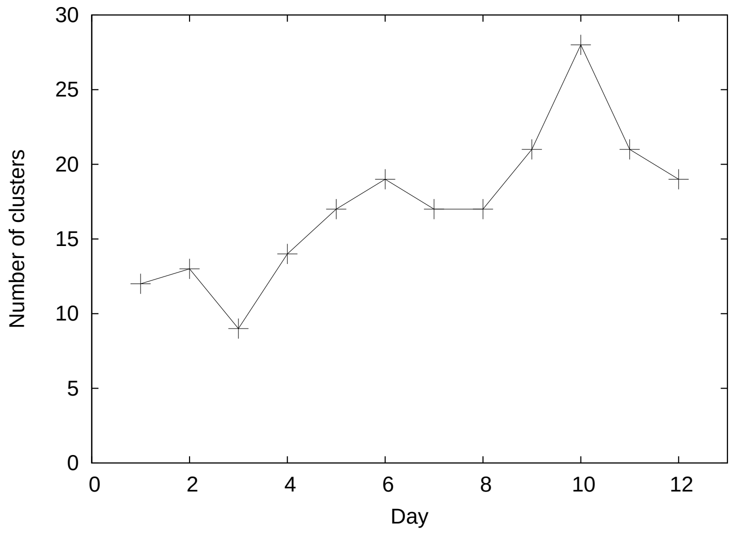


Figure 3.3. The number of clusters of the cumulative data set over the 12 days. The number of clusters is determined using 10 fold cross validation with expectation maximization clustering.

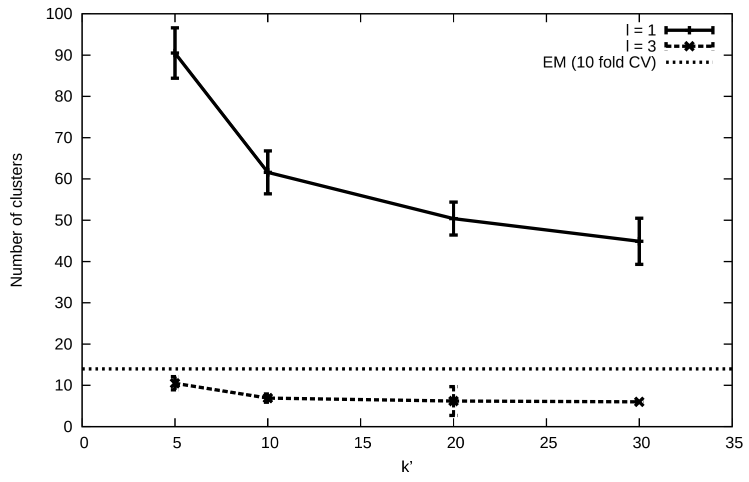


Figure 3.4. The mean and standard deviation of the number of clusters produced by the hybrid clustering algorithm. As expected, running the algorithm with a higher threshold value causes it to produce fewer clusters.

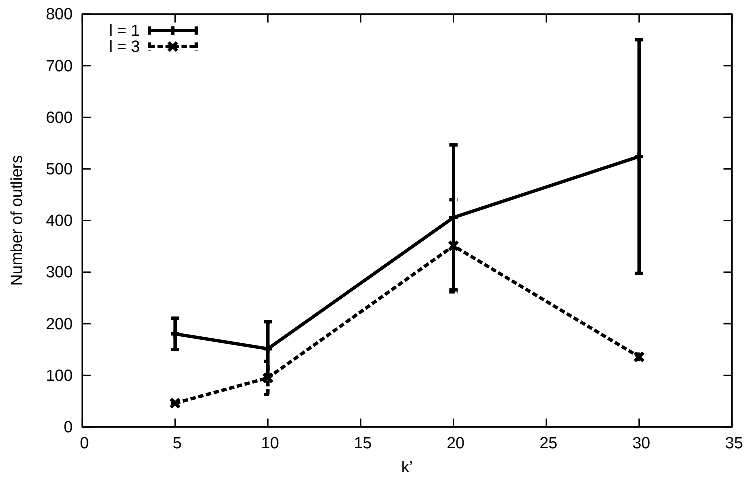


Figure 3.5. The mean and standard deviation of the number of outliers resulting from hybrid clustering. As expected, the running the hybrid algorithm with a high threshold value causes it to find fewer outliers.

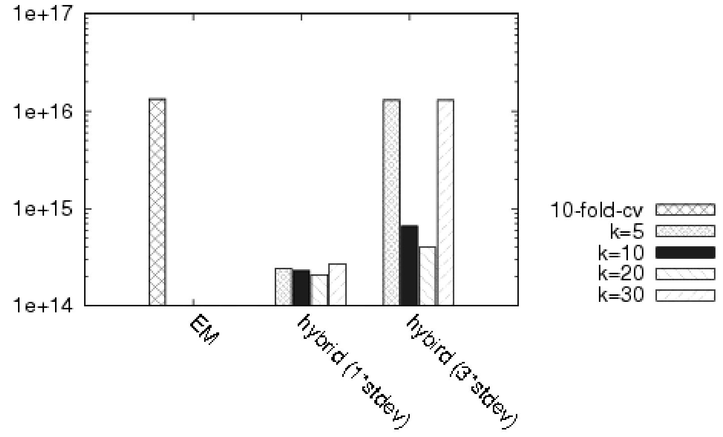
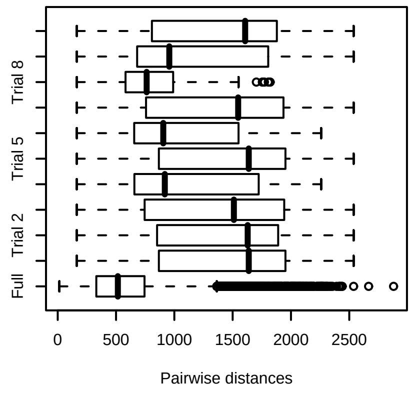
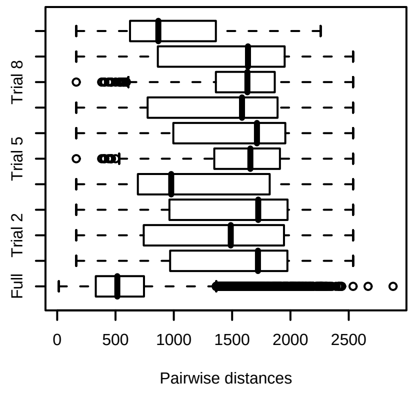


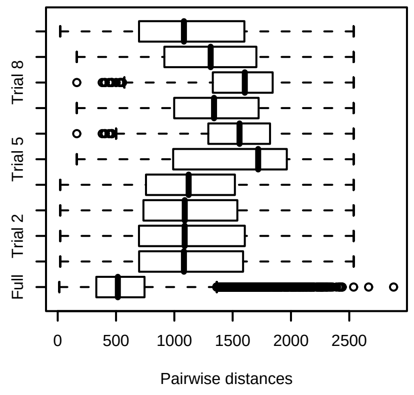
Figure 3.6. Sum squared error of the clusters for the expectation maximization and hybrid clustering algorithms. (Note the y axis is log-scale.) The hybrid algorithm produces clusters with less sum squared error than expectation maximization.



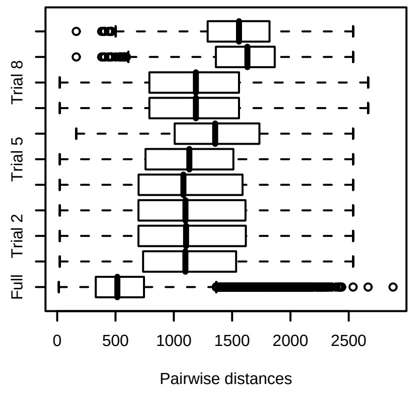
(a) $l = 1, k' = 5$



(b) $l = 1, k' = 10$

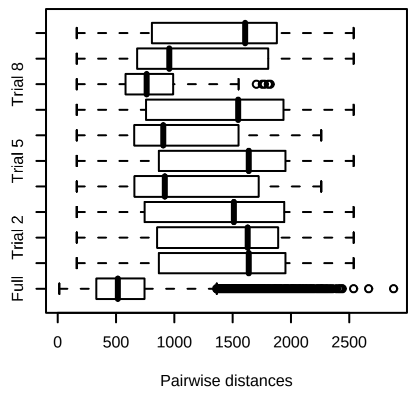


(c) $l = 1, k' = 20$

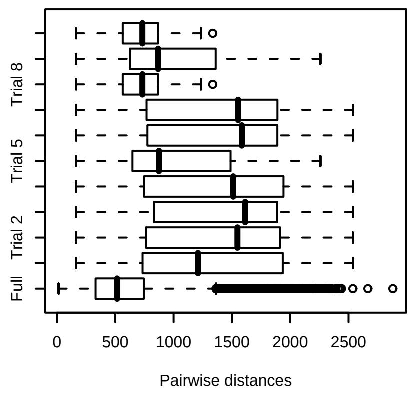


(d) $l = 1, k' = 30$

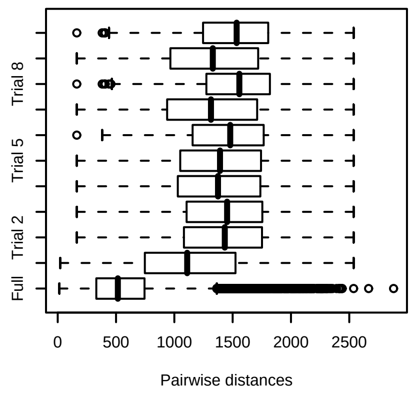
Figure 3.7. Box-plots of the distribution of distances between the outliers and their nearest neighbor in the full dataset for each trial of the hybrid clustering algorithm where $l = 1$. The bottom box-plot in each graph shows the distribution of distances from the nearest neighbor for all examples in the dataset.



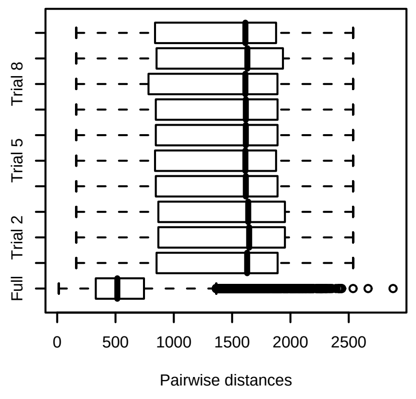
(a) $l = 3, k' = 5$



(b) $l = 3, k' = 10$



(c) $l = 3, k' = 20$



(d) $l = 3, k' = 30$

Figure 3.8. Box-plots of the distance between the outliers and their nearest neighbor in the full dataset for each trial of the hybrid clustering algorithm where $l = 1$. The bottom box-plot shows the distribution of distances from the nearest neighbor for all examples in the dataset.

CHAPTER 4

LINK SAMPLING FOR ANOMALY DETECTION IN STREAMING SOCIAL NETWORKS

4.1 ABSTRACT

Phone and email systems can produce social networks in a streaming fashion, which provides opportunities for the development of a variety of online applications, including emergency response management and organizational knowledge flow management. However, dealing with social networks that arrive as a stream of links is a difficult problem. We investigate link sampling from a graph for fast anomalous link detection in email and cell-phone networks. We use three methods for anomalous link detection—two neighborhood based and one path based—in conjunction with three methods of sampling suitable for streaming data. Each sampling method varies in the extent to which it considers the history of the stream. We evaluate the methods using Spearman's rank correlation and measure its degradation as the samples become smaller.

4.2 INTRODUCTION

Much work on social networks is constrained to only a sample of the full network. Traditionally, social network data has been gathered through interviews via two methods [26]:

- *Full network sampling* in which the connections between a predetermined set of individuals are mapped.

- *Snowball sampling* in which a sample is grown by mapping the connections of an individual in order to find additional individuals, whose connections are also mapped. This process continues until the network is large enough.

Technological networks, such as the world-wide web, email networks, or cell-phone networks, may be mapped more efficiently using software applications. These networks may be more complete; however, their size may make analysis difficult due to computational constraints.

Over the past several years, a number of fairly complete network datasets have become publicly available, including the Enron email dataset [32], the SourceForge Open Source Software dataset [3, 4], the Wikigraph dataset [25], and several scientific collaboration datasets from the arXiv e-print database [1, 69].

Social network data collected using surveys have the potential to contain more information about their links than technological networks; researchers can design surveys to collect more of the information relevant to their project. In contrast, the technological datasets contain only limited information about the links. Consider datasets such as the SourceForge network and the scientific collaboration network, where the fact that two people have contributed to a common project or are co-authors on a paper provides very little information regarding the nature of their interactions with each other. Email and cell-phone network datasets are more descriptive since they can provide more detail on the frequency of contact between two people; however, we must still be aware of the shortcomings of these types of datasets.

Grippa *et al.* [42], evaluate email networks as a tool for measuring knowledge flow in an organization. They compare an email network to a social network that consists of face-to-face, phone, and chat communications in addition to the email data. They found that email and phone interactions occur much less often than face-to-face and chat communications, which impacts the results of knowledge flow analysis. While it is important to recognize this issue, Frantz and Carley [38] show that three significant events at Enron translated into detectable changes in its email network.

In this paper we present work on the problem of fast anomaly detection in dynamic graphs. Our primary motivation for this work is an emergency response application, currently under development, which uses a cellular communication network as a sensor network. This work is also relevant where obtaining results from a large network dataset in a timely fashion is important. Examples of such applications include fraud detection, network intrusion detection, and organizational risk analysis.

We develop a method for quickly detecting anomalies in graphs with reasonably small space requirements. Since cell-phone networks tend to be large and transactions — records of SMS and phone calls — may arrive quickly, we begin by considering the data stream model. A data stream is a dataset, produced by some ongoing process, that arrives one item at a time, such as prices on a stock ticker [14]. Data streams tend to be so large that it is neither feasible to store the data in main memory nor repeatedly read the entire dataset from disk. The data stream model handles these characteristics by imposing the following constraints on algorithms for streaming data:

- Each item in the stream may only be read once.
- The items in the stream must be processed in the order in which they arrive.
- The algorithm should not use more than polylogarithmic space, with respect to the number of items in the stream.

Sometimes the model is relaxed, and a small number of passes over the data are permitted to reduce the memory requirements. This relaxation may be beneficial to applications such as database query processing where streams are produced by sequential access of massive files on secondary storage, but is not useful for processing streaming data from sensors, since these are potentially unbounded in length [14, 45].

Previous work on graph mining streaming data has focused on subgraph matching for fraud detection, community detection, or community analysis. Cortes *et al.* [33] use edge aggregation to handle streaming phone call transactions for fraud detection. Their method retains only the most active, and heavily weighted, outgoing links for each node in the network. They account for the dynamic nature of the graph by reducing edge weights over time via an exponential decay. Using a database of fraudulent activity and a measure of difference between subgraphs, the network is searched for new instances of fraud. This approach is not suitable for our work because we cannot assume that only the most active edges are relevant.

Coble *et al.* [30, 31] present online versions of SUBDUE which incrementally identify common substructures in graphs. The graphs are compressed by collapsing commonly occurring subgraphs. This approach does not allow the removal of outdated information without re-expanding the common subgraphs. This is problematic for the present applications because we don't necessarily want to consider the entire history, since the graph is dynamic.

Aggarwal and Yu [8] present an online method for summarizing dynamic networks that supports offline queries for measuring the change in communities over time. This summarization stores the initial graph along with incremental snapshots. Over time, storage is reclaimed by merging older snapshots, meaning the frequency of the graph snapshots decreases for older data. This approach essentially stores the entire graph in a space efficient way; which is necessary for the community analysis Aggarwal and Yu propose. However, given the size of the cell-phone networks, we prefer a more memory efficient solution.

4.2.1 Contributions

Our goal is to quickly summarize the communication patterns of the streaming network to identify areas of anomalous behavior. These may not necessarily correspond to a single community but likely contains portions of a number of communities. We characterize anomalous behavior in terms of the likelihoods of the edges in the graph; however, we must deal with the size and dynamic nature of the network. We apply stream sampling methods designed for non-relational datasets to three large, real-world social networks. We evaluate three sampling methods, each of which weights the history of the stream differently. We evaluate the performance of neighborhood and path based anomalous link detection methods on samples of varying size drawn from the social network.

4.2.2 Organization

The remainder of the paper is organized as follows. Section 2 reviews previous work related to graph sampling, stream sampling methods, and anomalous link detection. Section 3 describe how samples of the transaction stream are translated into social networks. Section 4 describes the datasets. Section 5 describes our experimental setup and results, and section 6 presents our conclusions and future directions for this work.

4.3 RELATED WORK

In this section, we discuss related work coupled with the two main components of our work — sampling from streaming social networks and anomaly detection. We end by juxtaposing the related work in the context of our work, highlighting our contributions.

4.3.1 Sampling

This section reviews previous work on sampling methods. The first part, section 2.1.1, discusses methods of sampling networks. These works are a good starting point, but are not sufficient for our application since they assume static networks. To the best of our knowledge, these methods have not yet been applied to maintaining updated summaries of dynamic graphs. The second part, section 2.1.2, discusses methods of sampling from streaming data; however, these methods are designed for non-relational data. Our work combines ideas from these two topics to build samples of dynamic graphs from streaming data.

4.3.1.1 Sampling Static Networks

Stumpf *et al.* [85] analyze the effect of node sampling on the degree distribution of scale-free graphs. Node sampling consists of selecting some number of vertices in the graph at random along with the edges among these vertices. They argue that observations made on a sampled network can only generalize to the full network if the degree distribution of the two networks belong to the same family of probability distributions. They show analytically that subgraphs of scale-free graphs are not scale free.

Lee *et al.* [56] empirically examine the effect of sampling graphs on four graph metrics: degree distribution exponent, betweenness centrality distribution exponent, assortativity, and clustering coefficient. They consider three methods for sampling from a network:

- *Node sampling*: some fraction of the vertices, along with the edges connecting these vertices, are selected uniformly at random.
- *Link sampling*: some fraction of the edges are selected uniformly at random.
- *Snowball sampling*: a vertex in the graph is selected uniformly at random and breadth first search is used to extract a subgraph of the desired size.

They found that, for the most part, sampling the graph affects these metrics consistently.

Research in computer networks evaluates a wider range of graph sampling methods, motivated by an interest in performing detailed protocol simulations on small, realistic networks [55]. Methods for generating such graphs include:

- *Deletion methods*: edges or vertices in a graph are randomly removed.
- *Contraction methods*: edges and vertices are randomly eliminated by merging neighboring vertices.
- *Exploration methods*: extracts subgraphs via breadth first search or depth first search.

The deletion and exploration methods are similar to the sampling methods in [56], though several variants of both are examined in [55].

4.3.1.2 Sampling Streams: Non-relational Data

In this section, we briefly discuss three methods of sampling from streaming data, particularly applicable to standard non-relational data: sliding window, uniform reservoir sampling, and biased reservoir sampling. Each method provides an "anytime" sample of the dataset, meaning that the sample is updated with the arrival of each data item, and it always conforms to its specified representation of the data. On one extreme, a sliding window consists of only the most recent data items in the stream. On the other extreme, a uniform reservoir sample retains each data items in the stream with the same probability. A biased reservoir sample is in the middle ground: more recent data items have a higher probability of being retained in the sample.

Sliding Window. A sliding window stores only the most recent data items in the streams. They may be either fixed in size, *i.e.* hold the last n items, or they may contain the data for some period of time, *i.e.* the last day.

Uniform Reservoir Sampling. Vitter [89] describes a one pass algorithm for extracting a uniform random sample of n items from an arbitrarily large data set. The algorithm populates the sample, called the reservoir, with the first n data items. Each new item is placed in the reservoir with a probability of n/N, where N is the number of data items that have been seen so far, in which case an item (selected uniformly at random) is ejected from the reservoir. At any time during the arrival of the stream, the reservoir contains a uniform random sample of the items seen so far.

Biased Reservoir Sampling. Aggarwal [6] presents a compromise between the sliding window and uniform reservoir sampling approaches. The algorithm described by Vitter is modified to store an exponentially biased reservoir sample. This algorithm always adds the new data item to the reservoir. The probability that the new item replaces an item in the reservoir is the fraction of the reservoir that is populated, in which case an item is selected for replacement uniformly at random. Let $p(r,t)$ be the probability that the rth item in the stream is in the sample after t items arrive. Aggarwal shows that for this algorithm, $p(r,t) \propto e^{-(t-r)/n}$.

4.3.1.3 Summary

We combine link sampling with each of the stream sampling methods to summarize dynamic graphs. Link sampling is the most natural approach since the edges automatically provide the relevant vertices in the graph. Node sampling is problematic since an additional mechanism is required for capturing the edges between the vertices in the sample. Feigenbaum *et al.* [37] show that building a breadth first search tree from a stream requires multiple passes, so snowball sampling is not possible for this application. Section 3 describes algorithms for building summaries of dynamic graphs.

4.3.2 Anomalous Link Detection

Rattigan and Jensen [76] define the anomalous link detection problem as the task of finding "surprising" edges in a graph. Anomalous links may be found by determining the likelihood of each edge in the graph; the edges with a likelihood below some threshold may be considered anomalous. We use methods of determining edge likelihoods from work on a closely related problem: link prediction [57].

Each method described below is a likelihood measure computed for each edge, (u, v), in the graph.

4.3.2.1 Neighborhood Based Methods

We use two neighborhood measures: common neighbors and Jaccard's coefficient. These are drawn directly from the link prediction literature [57]. Common neighbors, c, is simply the number of neighbors u and v share. Formally, let $\Gamma(u)$ be the set of vertices that are connected to u with an edge:

$$c = |\Gamma(u) \cap \Gamma(v)| \tag{4.1}$$

Jaccard's coefficient, J, is the probability that a neighbor of u or v is a neighbor of both u and v. Formally,

$$J = \frac{|\Gamma(u) \cap \Gamma(v)|}{|\Gamma(u) \cup \Gamma(v)|} \tag{4.2}$$

4.3.2.2 Path Based Methods

Rooted PageRank is a path based method that measures the probability of a random walk starting at u reaching v when the walk fails at each step with some probability, α [57].

4.3.2.3 Summary

We use the neighborhood method and Jaccard's coefficient as discussed above. However, we provide a modification to rooted PageRank do minimize the memory constraints (see section 5.1 for details).

4.4 SAMPLING STREAMING GRAPHS

Each method described in section 2.1.2 provides an "anytime" sample, *i.e.* the data in the sample always corresponds to its specified characteristics. For non-relational datasets, this means that the sample is alway usable; however, for our data we must build the graph from the sample. We only use one snapshot of the graph in this work; however, for most applications, it is necessary to periodically analyze the network. There are two possible approaches for updating dynamic graph summaries:

- Each snapshot of the network can be build from scratch using the state of the sample at any given time (see algorithm 2).

Algorithm 2 REBUILDGRAPH

```
Let $G' = (V', E')$ be the graph sample.
Let $\{t_1, t_2, \dots\}$ be the set of times at which the graph is analyzed.
Let $E_{(t_i,t_j)}$ be the set of transactions that arrive between time $t_i$ and $t_j$.
for all $t_i \in \{t_1, t_2, \dots\}$ do
    for all $e_i \in E_{(t_{i-1},t_i)}$ do
        UPDATESAMPLE($e_i$)
    end for
    $V' \leftarrow \emptyset$
    $E' \leftarrow \emptyset$
    for all $e_i = (v_x, v_y) \in E'$ do
        $V' \leftarrow V' \cup \{v_x, v_y\}$
        $E' \leftarrow E' \cup \{e_i\}$
    end for
    Compute a likelihood measure for each $e_i \in E'$.
end for
```

- We can maintain an anytime graph by adding and removing edges as the samples are updated (see algorithm 3).

We use unweighted, undirected graphs, and we are only concerned with vertices that have degree greater than zero. For each edge, (v_i, v_j), if either vertex is not already in the graph, we add it. We then add the undirected edge, represented as two directed edges, if it is not already present. We use AVL trees for storing both the vertices and the edges. In the worst case, we must add two vertices, v_i and v_j, and two directed edges, (v_i, v_j) and (v_j, v_i). Adding a vertex is simply an AVL tree insert. Adding a directed edge consists of finding the source vertex in the vertex tree and inserting the destination into the vertex's adjacency tree. Each of these operations requires $O(\log(|V|))$ time, where $|V|$ is the number of vertices in the sample. Since there are a constant number of these operations, the overall time required for inserting an undirected edge is $O(\log(|V|))$.

Now suppose we want to remove an edge, (v_i, v_j), from the graph. The first step is to remove the undirected edge. If either vertex has degree zero after the edge removal,

Algorithm 3 UPDATEGRAPH

```
Let $G' = (V', E')$ be the graph sample.
Let $\{t_1, t_2, \dots\}$ be the set of times at which the graph is analyzed.
Let $E_{(t_i,t_j)}$ be the set of transactions that arrive between time $t_i$ and $t_j$.
Let $d(v_i)$ be the degree of vertex $v_i$.
$V' \leftarrow \emptyset$
$E' \leftarrow \emptyset$
for all $t_i \in \{t_1, t_2, \dots\}$ do
    for all $e_i = \{e_1, e_2, \dots\}$ do
        UPDATESAMPLE($e_i$)
        if edge $e_i = (v_x, v_y)$ is added to the sample then
            $V' \leftarrow V' \cup \{v_x, v_y\}$
            $E' \leftarrow E' \cup \{e_i\}$
        end if
        if edge $e_j = (v_x, v_y)$ is evicted from the sample then
            $E' \leftarrow E' \setminus \{e_j\}$
            if $d(v_x) = 0$ then
                $V' \leftarrow V' \setminus \{v_x\}$
            end if
            if $d(v_y) = 0$ then
                $V' \leftarrow V' \setminus \{v_w\}$
            end if
        end if
        Compute a likelihood measure for each $e_i \in E'$.
    end for
end for
```

it is also removed. Since removing an item from an AVL tree takes $O(\log(|V|))$ time—like insertion—the overall time complexity for edge removal is also $O(\log(|V|))$.

Suppose we choose to build each snapshot from scratch. Let the sample store $|E|$ edges. In the worst case, $|E| = |V|$, and the time required to build the graph is $O(|E| \log(|E|))$.

Now suppose we choose to update the whenever the sample changes. In the worst case, a change to the graph requires an edge insertion and an edge removal, which takes $O(\log(|E|))$ time. The frequency at which the sample updates depends heavily on the type of sample used. Both sliding window and biased samples change with the arrival of each data item. Updating a sliding window always consists of an edge insertion and deletion; however, updating a biased sample does not alway require the edge removal. Recall that the probability of edge eviction from a biased sample is the fraction of the reservoir that is occupied; therefore, the rate at which edges are removed from the graph increases over time. In contrast, the probability that the uniform sample changes is $|E|/N$, where N is the number of data items seen so far. The uniform sample, therefore, is expected to change less often as the stream progresses.

The appropriate choice depends on the situation. If the stream arrives very quickly, it may not be feasible to update the graph for each change of the sample. This is especially true for sliding windows and biased samples, since the arrival of each data item changes the sample. Additionally, if the sample is very large or the network must be analyzed frequently, it may not be feasible to repeatedly build the graph from scratch.

4.5 DATASET

We evaluate the effect of graph sampling on various methods of anomalous link detection using three datasets. A text message (SMS) network and a call network are the primary focus of our work. We also examine the Enron email dataset [32] since it is widely used and publicly available. Since the Enron dataset is fundamentally different from our cell-phone network, it provides an external reference point that enhances our analysis.

The SMS and call networks are extracted from the transaction records of a cellular communication company; we use one day of data from these records for our evaluation. An examination of the degree distribution of these datasets reveals vertices with unrealistically high degrees (see figures 4.1 and 4.2). Certainly, it is not feasible for someone to call 100,000 different people or send SMS to 10,000 different people in a

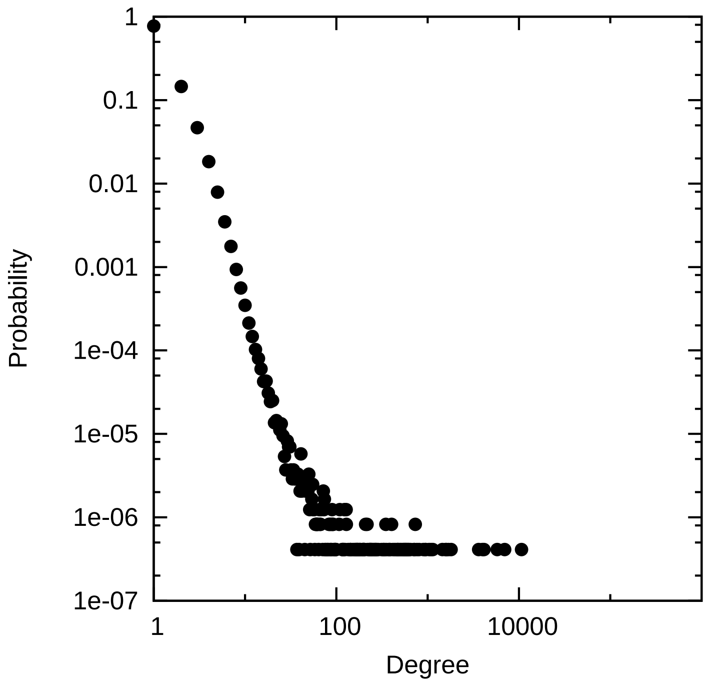


Figure 4.1. The degree distribution for the SMS dataset.

day, so we prune the vertices with the highest 0.1% degrees from these datasets. Such vertices may be present for a number of reasons. They may be services that distribute information to subscribers, or they may be fraudulent activity. These datasets contain only transactions initiated by subscribers of a single service provider. Any call made into the system from a non-subscriber that is not reciprocated does not appear as an edge in the graph.

The Enron email dataset is a snapshot of Enron's email server that was released by the Federal Energy Regulatory Commission [32]. We generate a social network from this snapshot by extracting the date, from, to, carbon-copy, and blind carbon-copy fields from each email. We use the `Date::Parse`[2] package for Perl to handle various timestamp formats found in the data. Several messages received unrealistic

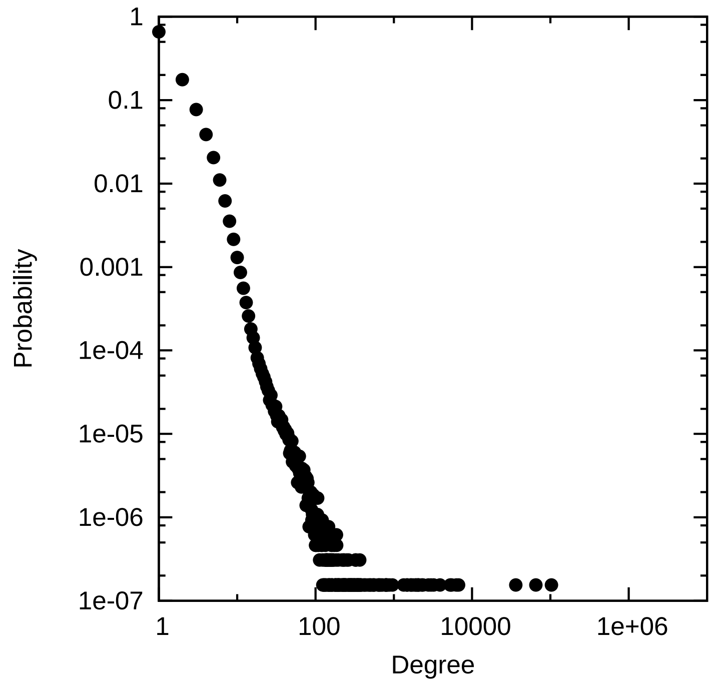


Figure 4.2. The degree distribution for the call dataset.

timestamps either due to noise in the data or failure of `Data::Parse` to handle certain timestamp formats. These data (emails supposedly sent in 1970 or 2020, for example) were removed. The dates in the final data set range from May 10, 1999 to January 31, 2002.

Table4.1 gives a brief summary of the social networks that are used in this paper. The vertices in the networks correspond to email address and phone numbers. The datasets are sets of transactions (SMS messages, phone calls, and emails) that we represent as undirected, unweighted graphs. Figures 4.1, 4.2, and 4.3 show the degree distributions of the SMS, call, and Enron datasets, respectively.

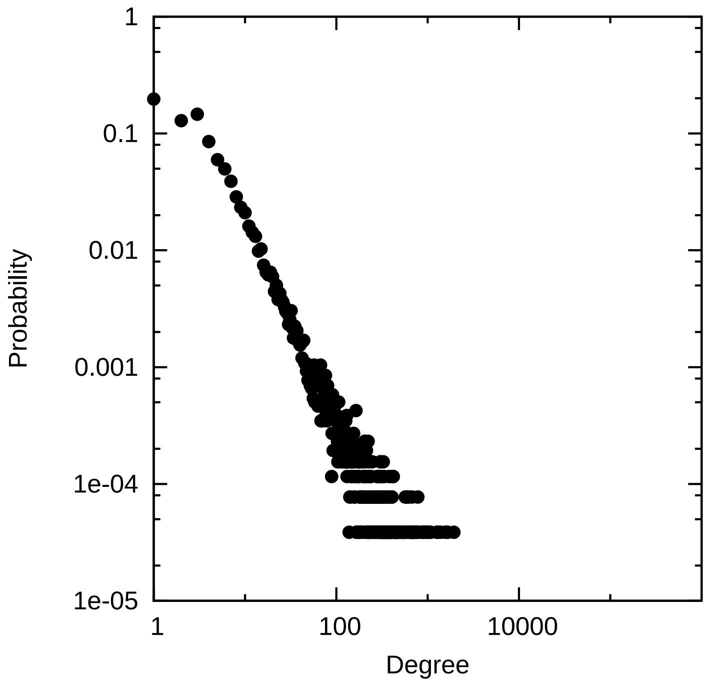


Figure 4.3. The degree distribution of the Enron dataset.

4.6 EXPERIMENTS

We examine how methods for anomalous link detection are affected by edge sampling from a graph. We use sample sizes ranging from 10% to 90% of the edges in the original graph. We expect the values of these measures to change as the graph is sampled, and we are interested in whether the anomalous edges in the full graph are also anomalous in the sample. Since larger common neighbors, Jaccard's coefficient, and rooted PageRank values are typically interpreted as higher edge likelihoods, we evaluate using Spearman's rank correlation. We compute the rank correlation only for the edges that appear in the sampled graph. Ideally, we would like to see high rank correlations, indicating that the edges remain in the same order of "anomalousness".

The implementations of common neighbors and Jaccard's coefficient are straight-

TABLE 4.1

GENERAL CHARACTERICS OF THREE NETWORK DATASETS

	vertices	transaction	edges
Enron	25,854	1,033,638	201,243
SMS (1 day)	2,350,793	3,339,708	1,597,818
Call (1 day)	6,261,633	8,019,290	5,243,128

forward; however, the normal approach for computing rooted PageRank is not feasible for the cell-phone datasets due to space constraints. In this section, we describe an approximation of rooted PageRank followed by a discussion of the results from our experiments.

4.6.1 Approximation of Rooted PageRank

Recall that rooted PageRank is the probability that a random walk starting at u reaches v when the walk fails at each step with some probability, α. The rooted PageRank of an edge can be drawn directly from the stationary distribution of the Markov Chain that represents the random walk described above [57]. A Markov chain is defined by a transition matrix that represents the probability of moving between any two vertices in a single step. Multiplying the matrix by itself $\mathbf{m}$ times produces a matrix that gives the probability of moving between any two vertices in exactly $\mathbf{m}$ steps. A transition matrix $\mathbf{R}$ has a stationary distribution, $\mathbf{R}^{(\mathbf{m})}$, when $\mathbf{R}^{(\mathbf{m})} = \mathbf{R}^{(\mathbf{m-1})}$ [64].

Unfortunately, it is not feasible to store a transition matrix for the SMS and call graphs in main memory due to their large number of vertices. Instead, we approximate the rooted PageRank using Monte Carlo trials. The Rooted PageRank is estimated to be the fraction of Monte Carlo trials in which the random walk reaches the destination. Since the walk fails at each step independently and with a known probability, α, we can bound the length of the walk using a geometric distribution. We bound the walk by halting, for some threshold, θ, when the probability of reaching step n is less than θ. We halt, and fail, when the walk reaches step

$$n = \log_{1-\alpha} \frac{\theta}{\alpha} + 1 \tag{4.3}$$

Note that we remove the edge (u, v) from the graph before running the random walks and re-add it once the rooted PageRank has been approximated for the edge.

Given this value, we estimate the number of paths of length n based on the average degree of the graph, $\bar{d}$, to be $n^{\bar{d}}$. The number of Monte Carlo trials is proportional to the estimated number of paths. For our experiments, we set $\alpha = 0.15$ and $\theta = 0.001$, so $n \approx 31$, and we run $100n^{\bar{d}}$ trials. Table 4.2 shows the average degree, $\bar{d}$, and the estimated number of paths, $n^{\bar{d}}$, for each dataset. This approach is not feasible for the Enron dataset, due to its high average degree. Computation of rooted PageRank for the SMS and call datasets is feasible, though time-consuming.

4.6.2 Results and Discussion

Figures 4.4, 4.5, and 4.6 show the rank correlations described above for the three datasets. Interestingly, the rank correlations of all three methods on the Enron dataset follow roughly the same trend; though using a uniform sample in conjunction with common neighbors and Jaccard's coefficient performs better than a biased sample or sliding window. This may be due to the time coverage of this dataset: it extends two months beyond the large scale layoffs coinciding with the company's bankruptcy announcement

TABLE 4.2

CHARACTERISTICS OF NETWORK DATASETS RELATED TO ROOTED PAGERANK

	$\bar{d}$	$n^{\bar{d}}$	$\|r_{u,v}\|$
Enron	15.58	1.72×10^{23}	6.68×10^{8}
SMS	1.36	1.06×10^{2}	5.63×10^{12}
Phone	1.68	3.20×10^{2}	3.92×10^{13}

on December 2, 2001, which likely had a significant impact on the company's email network (see figure 4.7) [38]. The biased sample and sliding window rely heavily on recent information and, for small sample sizes ignore, all edges in the graph appearing before December 2, 2001. The uniform sample, on the other hand, takes into account the whole history, and, therefore, better captures the characteristics of the entire dataset.

In contrast, using a sliding window results in the best rank correlations on the call dataset for Jaccard's coefficient for small samples. This higher rank correlation may be due to the fact that the portion of the dataset the sliding window relies on — the last transactions of the day — is also the most active (see figure 4.8 for the volume of call transactions throughout the day). As the sample sizes become larger, the sliding window, like the uniform and biased samples, includes more transactions from earlier in the day. This would explain why all three sampling methods perform comparably for larger samples. We see a similar phenomenon, though to a lesser extent, on the SMS dataset. This may be due to the consistency of the activity volume after noon and the smaller peak near the end of the day (see figure 4.9).

These results make sense if we think about how people may use their phones. For example, if people tend to call their friends in the evening, after work (note that in figure 4.8 there is a peak around five o'clock in the evening), perhaps to make evening plans or to visit with distant loved ones, the social network is described more accurately by only these transactions, which roughly correspond to the last transactions of the day. In this case, a uniform sample is likely to impact highly active and descriptive transaction sub-streams more severely, which in turn degrades the performance of the overall anomalous link detection approach.

The rank correlations are high across the board for the common neighbors measure on the cell-phone network datasets. Issues related to the sparsity of the datasets raise questions about how much meaning can be attached to these results. Most of the edges in the cell-phone network datasets ($> 85\%$) have the worst possible value (no common neighbors). The vertices connected by these edges have no common neighbors in any sample in which they appear (removing edges from a graph never increases the number of common neighbors for a pair of vertices). Since these edges are in the majority, their values for these measures do not change for any sample in which they are included. Since they are likely to dominate any sample, the high rank correlations for common neighbors seem to be trivial results.

4.7 CONCLUSIONS

In this paper, we evaluate the impact of edge sampling a graph on anomalous link detection. We use three real-world social networks which vary widely in size and density. We evaluate three methods of computing edge likelihood: common neighbors, Jaccard's coefficient, and rooted PageRank. Rooted PageRank outperforms Jaccard's coefficient in terms of Spearman's rank correlation; however, it is also much more computationally expensive. Most of the results for common neighbors are not meaningful.

We investigate three methods of sampling, over a range of sample sizes, using Spearman's rank correlation. Among the these methods, there is no clear winner; though we make some observations. For Jaccard's coefficient:

- The sliding window performs best on small samples of the rapidly changing cell-phone network graphs.
- The uniform sample performs best on small samples of the Enron email dataset, which evolves more gradually.

For rooted PageRank on the cell-phone network graphs:

- The sliding window performs consistently well compared to the other samples.
- The uniform sample performs worst on small samples.
- The biased sample performs worst on large samples.

The fact that the rank correlations for all trials are positive is promising; however, the extent to which these values decline for small samples is a concern.

4.8 PROPOSED RESEARCH

In this chapter we examine methods for reducing relational datasets that arrive as a stream for the fast detection of anomalous links. We currently focus only on a single snapshot of the full dataset and propose to investigate the changes in the common neighbors, Jaccard's coefficient, and rooted PageRank distributions over time for both the full data set and samples of the datasets described above. We will also examine various subsets of the full network by extracting transactions for a diverse set of cities.

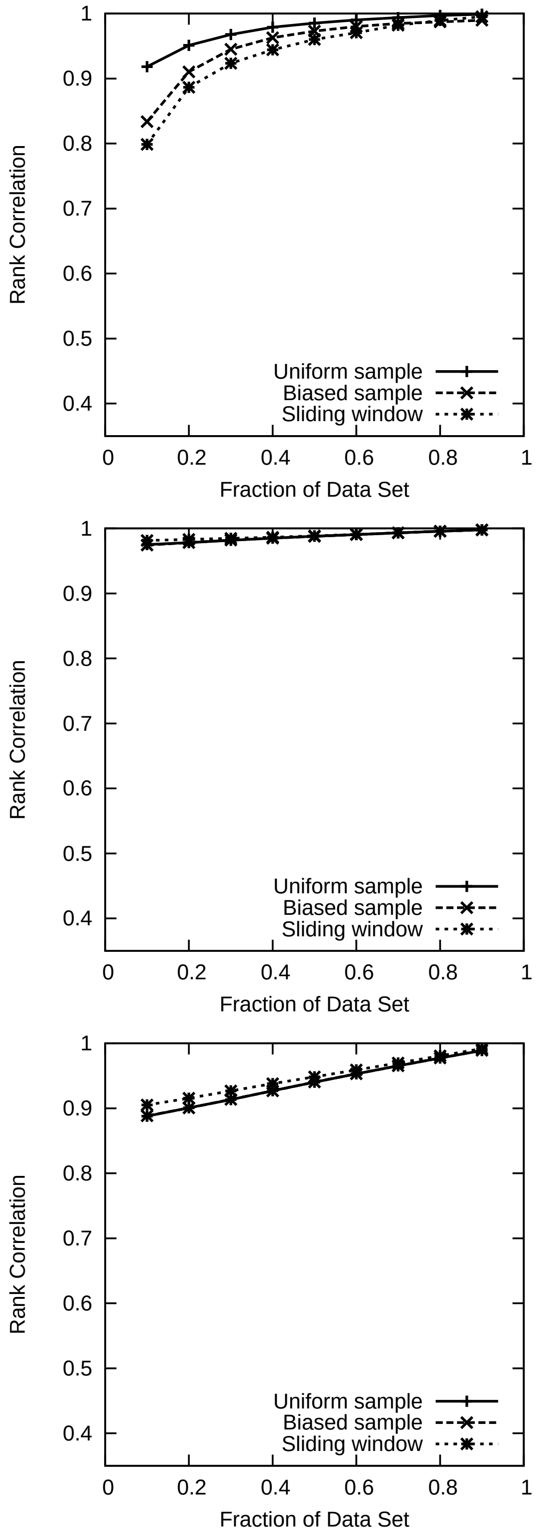


Figure 4.4: The rank correlation for common neighbors on the Enron (left), SMS (center), and call (right) datasets for varying sample sizes.

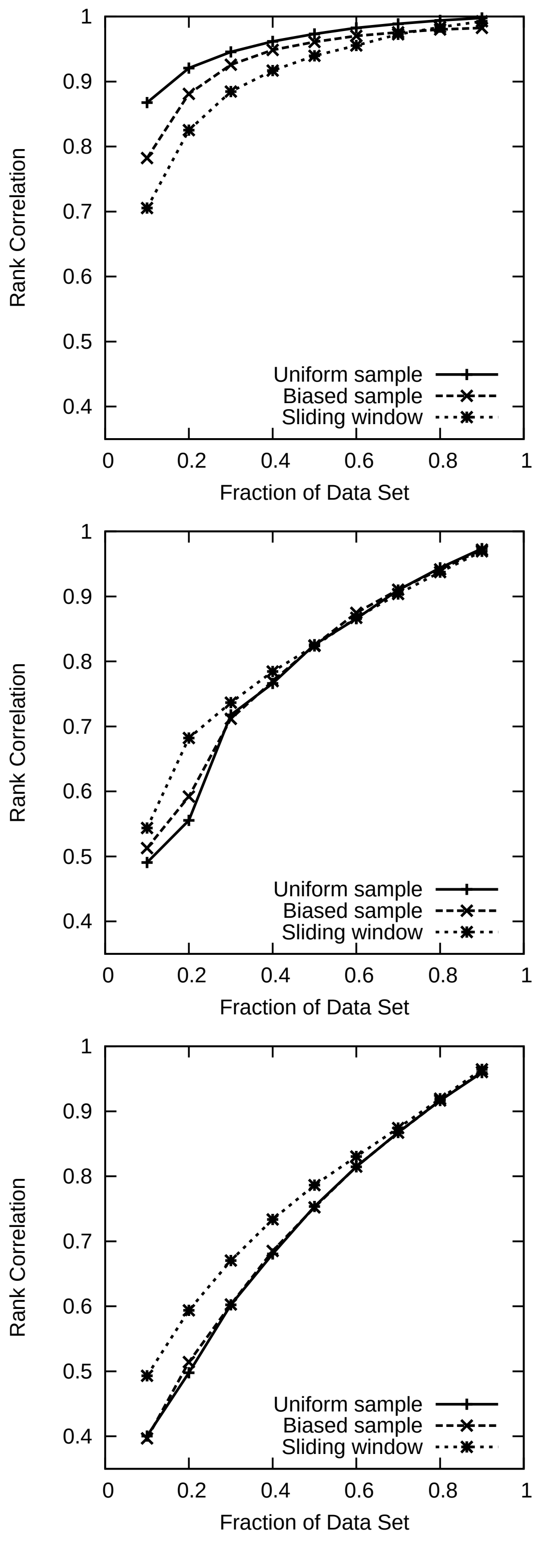
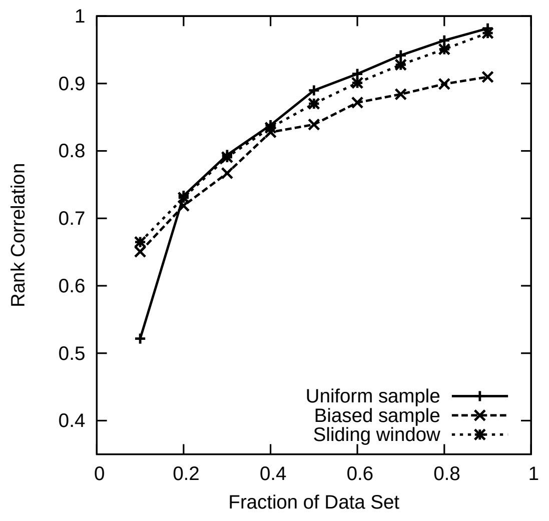


Figure 4.5: The rank correlation for Jaccard's coefficient on the Enron (left), SMS (center), and call (right) datasets for varying sample sizes.



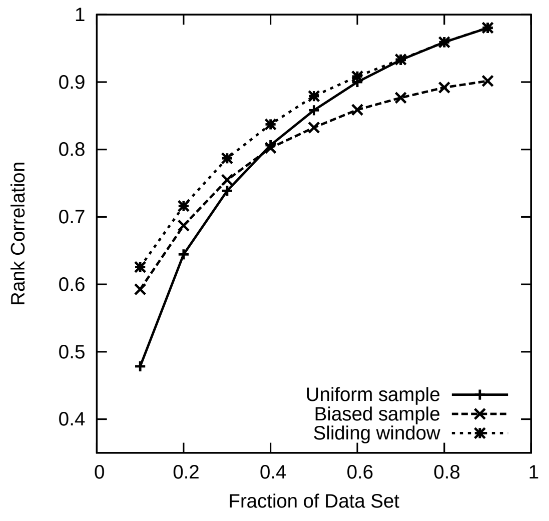


Figure 4.6. The rank correlation for rooted PageRank on the SMS (top) and call (bottom) datasets for varying sample sizes. It was not feasible to compute rooted PageRank on the Enron dataset using our approximation (see section 4).

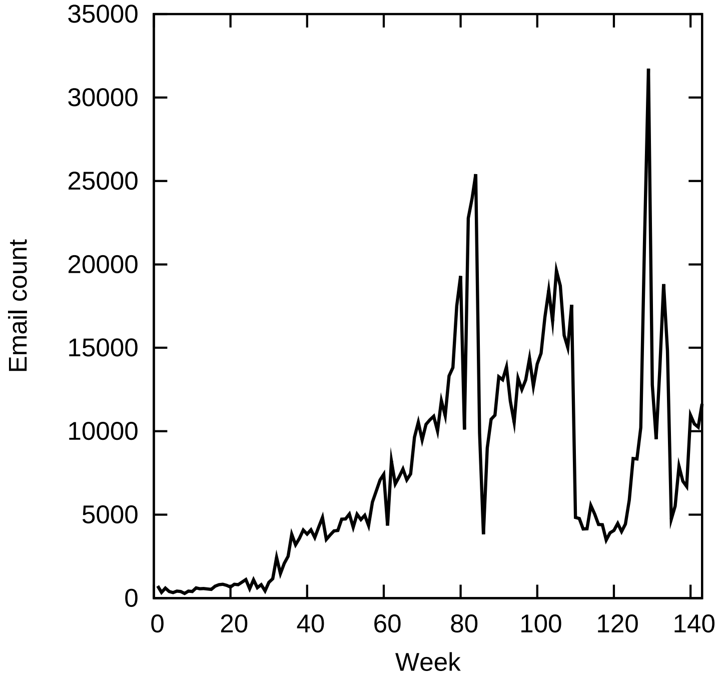


Figure 4.7. The volume of email sent by week from May 10, 1999 to January 31, 2002.

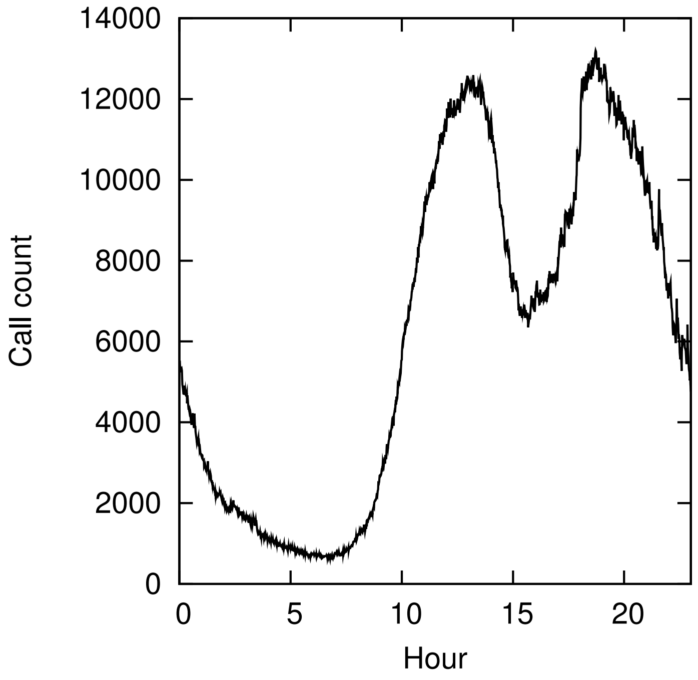


Figure 4.8. The volume of calls made over a day.

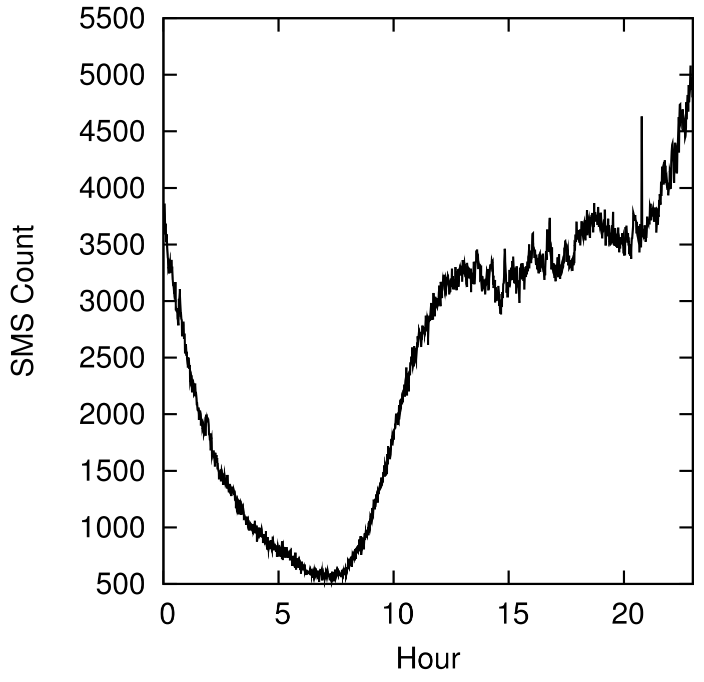


Figure 4.9. The volume of SMS messages sent over a day.

CHAPTER 5

A REAL-TIME DATA SOURCE PROTOTYPE

In this chapter, we present an overview of the real-time data source (RTDS). We discuss requirements, describe a prototype implementation and summarize lessons learned, and list implementation issues and proposed research relating to this component.

5.1 OVERVIEW OF THE REAL-TIME DATA SOURCE

The real-time data source will receive raw transaction data from a cellular service provider and serve requests for streaming data from other components of the WIPER system, namely the Detection and Alert System, the Simulation and Prediction System, and the Decision Support System. These components may request the transaction data or aggregated summaries of the data, *i.e.* a summary consisting of the number of transactions (calls, SMS, *etc*), in a series of time intervals. Some clients may only require data from a limited geographical area, so the system will provide this filtering functionality. The system will provide a uniform stream request interface for all clients and must support many clients simultaneously. In particular, the Simulation and Prediction System may contain many clients as it runs ensembles of simulations utilize new data as it becomes available.

The remainder of this section provides a detailed description of the real-time data source.

5.1.1 Incoming Data Stream

The incoming data from the cellular providers will consist of a stream of transactions. Each transaction consists of 6 fields (see figure 5.1):

1. the date on which the date on which the transaction occurs,
2. the time at which the transaction occurs,
3. the anonymized ID of the person initiating the transaction,
4. the anonymized ID of the person receiving the receiving the transaction,
5. the ID of the tower through which the transaction was initiated or received, and
6. the transaction type which whether the transaction is a phone call or SMS message and whether the transaction is generated by the initiating or receiving handset.

```
   1        2       3      4            5             6
20061101,203311,596887,2295,214031250124834.000,SOM
```

Figure 5.1. The transaction format.

5.1.2 Input/Output

The RTDS will receive stream requests from clients via web services using SOAP remote procedure calls. Data streams will be transmitted to the clients over UDP sockets.

5.1.3 Outgoing Data Stream

The RTDS will provide two types of streaming data to its clients.

- Raw transaction streams simply forward transactions to clients as they are received by the RTDS.
- Summary transaction streams contain the number of transactions per client specified time interval.

Both of these streams may be filtered by tower ID. A whitelist is provided by the client and any transaction containing a tower ID that is not in the whitelist is ignored.

5.1.4 Temporal Constraints

The RTDS is driven by incoming transactions from the cellular service provider. These tasks are neither periodic nor sporadic; however, we can establish an upper bound on the rate at which transactions arrive. For this reason, the system will follow the rate-based execution model.

5.2 PROTOTYPE IMPLEMENTATION

Our prototype for the real-time data source is implemented in Ruby, an interpreted, object-oriented scripting language, and uses a periodic task model. Ruby was originally selected for its simple, easy to use web services classes, though it also provides support for multi-threading with a large priority space.

5.2.1 The Task Set

There are two tasks associated with each client receiving a summary stream.

- An aperiodic task runs whenever a new transaction arrives from the service provider. This task updates the summary for the current time interval.
- A periodic task runs at the end of each interval. It sends the interval summary to the client and resets the summary for the next interval.

5.2.2 Task Scheduling

We use the periodic task model for the prototype and only concerned ourselves with temporal constraints within the RTDS; that is, we assume no latency from the service provider to the data source, and we ignore the latency between the data source and clients altogether.

Let the delay be the time between the end of the of the interval to the time the summarized and filtered data is sent out to the clients. Ideally, the maximum delay will be the length of the period of the shortest periodic task, though delays of the task's period may be tolerated. This requirement aims to provide both absolute and relative temporal consistency since it establishes both a maximum age and a maximum difference in ages for data distributed to the clients at the end of any given interval. Additionally, this constraint guarantees that the system keeps up with the data: if the time required to summarize an interval (after all the data for that interval was received) was longer than any given interval, the quality of the data, due to perish, would decrease as time passes.

We implement and evaluate three standard scheduling algorithms for the periodic tasks: deadline monotonic scheduling, earliest deadline first, and weighted fair queuing. This task set differs from more traditional periodic real-time tasks sets in that the aperiodic tasks take on greater importance. One approach for catering to aperiodic tasks is the deadline monotonic scheduling algorithm [12] which is a generalization of the rate monotonic algorithm that allows for periodic task deadlines to be before the end of the task's period. We define the soft deadline for each task to be the minimum period in the task set. The hard deadline is the end of the task's period.

5.2.3 Utilization Analysis

The utilization of a real-time system with only periodic tasks is computed by

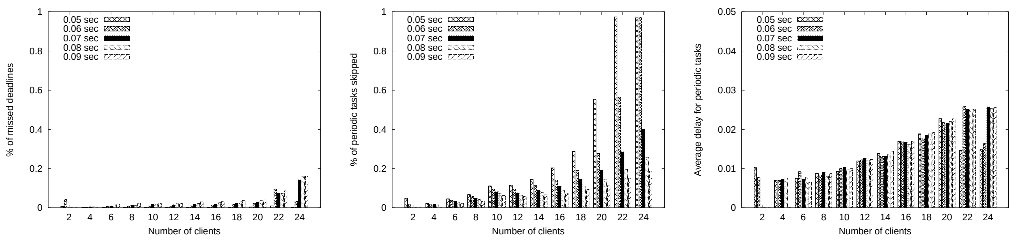
$$U = \sum_{i=1}^{n} \frac{C_i}{T_i} \tag{5.1}$$

where C_i and T_i are the worst case execution time and the period, respectively, of periodic task i. Let P be the worst case execution time of the periodic jobs. Since the periodic tasks depend on the aperiodic tasks for correctness, we also must consider the aperiodic tasks when computing the utilization. Let A be the worst case execution time of the aperiodic jobs and R be the upper bound of the rate at which the transactions arrive at the data server. RT_i gives us an upper bound on the number of aperiodic tasks that run in the period. The utilization is then

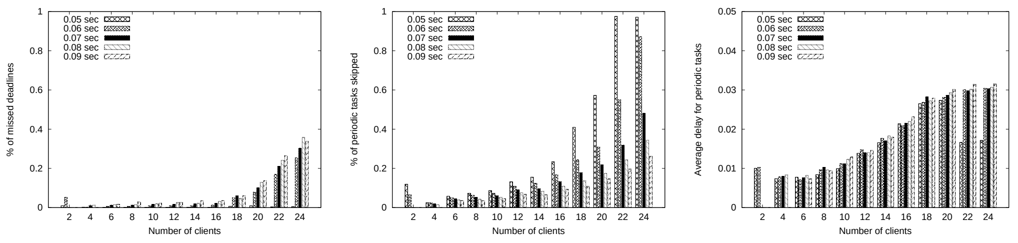
$$\sum_{i=1}^{n} \frac{P + RT_iA}{T_i} \tag{5.2}$$

Since we stipulate that all periodic jobs must execute within the shortest interval, we also add the following constraint to the schedulability analysis:

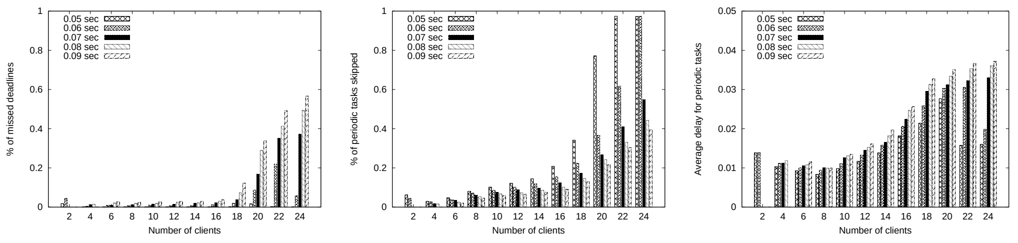
$$\sum_{i=1}^{n} P \leq \mathsf{min}(T_i). \tag{5.3}$$



(a) DMS



(b) EDF



(c) WFQ

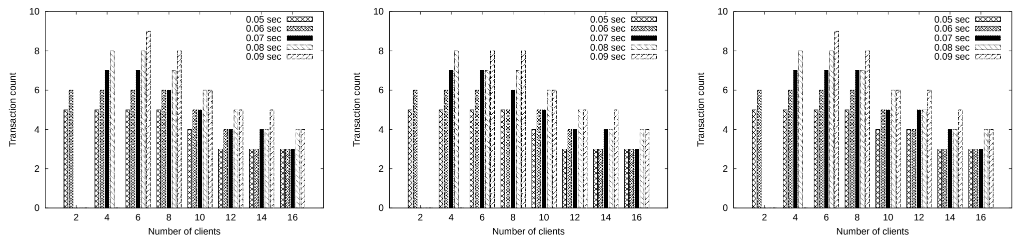
Figure 5.2: The percent of missed deadlines, percent of skipped periodic tasks, and the average delay of the periodic tasks for an increasing number of clients with a task set consisting of periods, $0.05, 0.06, 0.07, 0.08$, and 0.09 seconds.

5.2.4 Evaluation and Results

The RTDS runs on a dual Pentium III (650 MHz) with 1 GB of main memory; however, since Ruby threads are contained within the Ruby interpreter process, the RTDS can only make use of one processor.

We evaluate the three algorithms based on four measures:

- the rate at which periodic tasks miss their soft deadlines (in practice, hard deadlines were rarely missed, rather tasks were skipped),
- the rate at which periodic tasks are skipped due to being released after the end of



(a) DMS (b) EDF (c) WFQ

Figure 5.3: The median result sent to the clients with a task set consisting of periods $0.05, 0.06, 0.07, 0.08$, and 0.09 seconds.

their period,

- the average delay for periodic tasks, *i.e.* the average time between the end of the summary interval and the time the data is sent to the client, and
- the median output of the periodic tasks for each interval.

Figure 5.2 shows the rate of missed deadlines, rate at which jobs are skipped, and the average delay for the deadline monotonic, earliest deadline first, and weighted fair queuing scheduling algorithms. Figure 5.3 the median output of the periodic tasks for each interval. The task sets consist of 2 to 24 periodic tasks (clients) with periods of $0.05, 0.06, 0.07, 0.08,$ and 0.09 seconds. Figure 5.2 shows that the deadline monotonic scheduling algorithm performs significantly better in terms of percentage of missed deadlines than either of the other two algorithms and slightly better in terms of average delay when there are more than 16 clients. All three algorithms perform comparably in terms of the fraction of jobs skipped due to being released after the hard deadline and in terms of the server correctness shown in figure 5.3.

Table 5.1 shows empirically measured maximum execution times for both periodic and aperiodic tasks in this system. Based on these execution times and the equation derived in section 5.2.3 for system utilization (equation 5.2), the processor utilization is only about 0.26 when the correctness begins to degrade at approximately eight clients.

TABLE 5.1

TASK EXECUTION TIME

Task Type	Execution Time
periodic	1.6×10^{-3} seconds
aperiodic	8.5×10^{-5} seconds

5.2.5 Discussion

The fact that the system fails with a utilization of only 0.26 is a significant problem. We believe that our choice of programming language contributes substantially to this; therefore, we will migrate the system to a lower level compiled language, C. Since the system is driven by aperiodic behavior, the arrival of tasks from the service provider, we will abandon the periodic task model for rate-based execution.

5.3 CHALLENGES AND PROPOSED RESEARCH

The main challenge we face is a lack of control over the actual network infrastructure that delivers data to the real-time data source. While using a cell-phone network eliminates the effort and expense of deploying a sensor network, it imposes a pre-designed data collection mechanism over which we have no control. Most notably we have no control over the routing protocol, so we cannot guarantee highest priority to older packets that must travel further. This issue may result in out-of-order transaction arrival, a detailed we ignored in developing the prototype. The extent of this problem may not be known at the time of deployment and will change over time, so the system must be adaptive. We propose to develop a method for dynamically determining the maximum latency in order to balance delay times with omitted data due to late arrival.

CHAPTER 6

SUMMARY AND CONCLUSION

6.1 SUMMARY OF PROPOSED RESEARCH

We propose the following research:

- Develop a streaming hybrid clustering algorithm that requires no *a priori* knowledge of the number of clusters.
- Identify feasible methods for reducing graph data for fast analysis.
- Identify graph features that can be quickly computed and allow the identification of anomalous behavior in graphs.
- Develop a real-time system, following the rate-based execution model, that receives out-of-order streaming data and provides streaming data (either raw or summarized) to clients while performing on-line balancing between missing data due to jitter and data propagation delay.

6.2 PROPOSED SCHEDULE

For the purposes of scheduling, we divide the proposed research into three areas: traditional stream mining, relational stream mining, and real-time systems.

- The traditional stream mining work is the most mature. We have a conference publication [72] and a journal publication (to appear) [71] based on this work. We believe that the next iteration of this work will be ready for conference submission early in 2008 and journal submission in late 2008.

- The relational stream mining work consists of two parts: reducing graphs for fast analysis and detection of anomalous behavior in graphs. Work on the first component will be submitted to SIAM's data mining conference in early October 2007. We expect conference level work for the second component to be ready early in 2008. We expect journal submissions on both components in late 2008 or early 2009.

- The real-time systems component is the least mature. Given the drawbacks of the prototype, the real-time data source must be completely redesigned and rebuilt. We expect to have a conference submission ready in mid 2008 and a journal submission in early 2009.

We expect that this work will culminate in a dissertation that will be ready for defense in March 2009.

BIBLIOGRAPHY

1. ArXiv. http://www.arxiv.org.

2. Date::Parse. http://search.cpan.org/˜gbarr/TimeDate-1.16/lib/Date/Parse.pm.

3. OSS research portal. http://zerlot.cse.nd.edu.

4. Sourceforge.net. http://sourceforge.net.

5. Charu C. Aggarwal. A framework for diagnosing changes in evolving data streams. In *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, pages 575–586, 2003.

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

7. Charu C. Aggarwal, Jiawei Han, Jianyong Wang, and Phillip S. Yu. A framework for clustering evolving data streams. In *Proceedings of the 29th VLDB Conference*, 2003.

8. Charu C. Aggarwal and Philip S. Yu. Online analysis of community evolution in data streams. In *Proceedings of the ACM SIAM Conference on Data Mining*, 2005.

9. Réka Albert and Albert-László Barabási. Statistical mechanics of complex networks. *Reviews of Modern Physics*, 74:47–97, 2002.

10. Réka Albert, Hawoong Jeong, and Albert-László Barabási. Diameter of the world-wide web. *Nature*, 401:130, 1999.

11. Tracking cell phones for real-time traffic data. Associated Press, 2005.

12. N. C. Audsley, A. Burns, M. R. Richardson, and A. J. Wellings. Hard real-time scheduling: The deadling monotonic approach. In *Proceedings 8th IEEE Workshop on Real-Time Operating Systems and Software*, 1991.

13. Stefan Axelsson. Intrusion detection systems: A taxonomy and survey. Technical Report TR-99-15, Department of Computer Engineering, Chalmers University of Technology, 2000.

14. Brian Babcock, Shivnath Babu, Mayur Datar, Rajeev Motwani, and Jennifer Widom. Models and issues in data stream systems. In *Proceedings of the 21st ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, pages 1–16, June 2002.

15. Albert László Barabási. *Linked: The New Science of Networks*. Perseus Publishing, 2002.

16. Albert-László Barabási and Réka Albert. Emergence of scaling in random networks. *Science*, 286:509–512, October 1999.

17. Vic Barnett and Toby Lewis. *Outliers in Statistical Data*. John Wiley and Sons, 1994.

18. Sanjoy Baruah and Nathan Fisher. Real-time scheduling of sporadic task systems when the number of distinct task types is small. In *Proceedings of the 11th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, 2005.

19. Sanjoy K. Baruah, Aloysius K. Mok, and Louis E. Rosier. Preemptively scheduling hard-real-time sporadic tasks on one processor. In *Proceedings of the 11th IEEE Real-Time Systems Symposium*, 1990.

20. Salvatore Belardo, Kirk R. Karwan, and William A. Wallace. Managing the response to disasters using microcomputers. *Interfaces*, 14(2):29–39, March-April 1984.

21. Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, 1975.

22. C.A. Bicking and Frank M. Gryna, Jr. *Quality Control Handbook*, chapter Process Control by Statistical Methods, pages 23–1—23–35. McGraw Hill, 1979.

23. S. Boccaletti, V. Latora, Y. Moreno, M. Chavez, and D.-U. Hwang. Complex networks: Structure and dynamics. *Physics Reports*, 424:175–308, 2006.

24. Richard J. Bolton and David J. Hand. Statistical fraud detection: A review. *Statistical Science*, 17(3):235–255, 2002.

25. Luciana S. Buriol, Carlos Castillo, Debora Donato, Stefano Leonardi, and Stefano Millozzi. Temporal analysis of the wikigraph. In *IEEE/WIC/ACM International Conference on Web Intelligence*, 2006.

26. Deepayan Chakrabarti and Christos Faloutsos. Graph mining: Laws, generators, and algorithms. *Computing Surveys*, 38:2, March 2006.

27. Amitabh Chaudhary, Alexander S. Szalay, and Andrew W. Moore. Very fast outlier detection in large multidimensional data sets. In *Proceedings of the ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery*, 2002.

28. Eng Yeow Cheu, Chee Keongg, and Zonglin Zhou. On the two-level hybrid clustering algorithm. In *International Conference on Artificial Intelligence in Science and Technology*, pages 138–142, 2004.

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

30. Jeffrey Coble, Diane J. Cook, and Lawrence B. Holder. Structure discovery in sequentially-connected data streams. *International Journal on Artificial Intelligence Tools*, 15(6):917–944, December 2006.

31. Jeffrey A. Coble, Runu Rathi, Diane J. Cook, and Lawrence B. Holder. Iterative structure discovery in graph-based data. *International Journal on Artificial Intelligence Tools*, 14:101–124, 2005.

32. William W. Cohen. Enron email dataset. http://www.cs.cmu.edu/˜enron.

33. Corinna Cortes, Daryl Pregibon, and Chris Volinsky. Computational methods for dynamic graphs. *Journal of Computational and Graphical Statistics*, 12:950–970, 2003.

34. Frederica 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.

35. Dorothy E. Denning. An intrusion-detection model. *IEEE Transactions on Software Engineering*, 13(2):222–232, 1987.

36. Tom Fawcett and Foster J. Provost. Adaptive fraud detection. *Data Mining and Knowledge Discovery*, 1:291–316, 1997.

37. Joan Feigenbaum, Sampath Kannan, Andrew McGregor, Siddharth Suri, and Jian Zhang. Graph distances in the streaming model: The value of space. In *Proceedings of the 16th Symposium on Discrete Algorithms*, 2005.

38. Terrill L. Frantz and Kathleen M. Carley. The dynamics of organizational chatter. In *Proceedings of the North American Association for Computational Social and Organization Science (NAACSOS)*, 2006.

39. 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 *Proceedings of the International Conference on Computational Science*, 2006.

40. Mark Gaynor, Margo Seltzer, Steve Moulton, and Jim Freedman. A dynamic, data-driven decision support system for emergency medical services. In *Proceedings of the International Conference on Computational Science*, 2005.

41. Lise Getoor and Christopher P. Diehl. Link mining: a survey. *SIGKDD Explorations Newsletter*, 7(2):3–12, 2005.

42. Francesca Grippa, Antonio Zilli, Robert Laubacher, and Peter A. Gloor. E-mail may not reflect the social network. In *Proccedings of the North American Association for Computational Social and Organization Science (NAACSOS)*, 2006.

43. Supito Guha, Adam Meyerson, Nina Mishra, Rajeev Motwani, and Liadan O'Callaghan. Clustering data streams: Theory and practice. *IEEE Transactions on Knowledge and Data Engineering*, 3:515–528, May/June 2003.

44. John A. Hartigan. *Clustering Algorithms*. Wiley Series in Probability and Mathematical Statistics. John Wiley & Sons, 1975.

45. Monika Rauch Henzinger, Prabhakar Raghavan, and Sridar Rajagopalan. Computing on data streams. Technical Report 1998-001, Digital Systems Research Center, 130 Lyttono Avenue, Palo Alto, CA, May 1998.

46. Victoria J. Hodge and Jim Austin. A survey of outlier detection methodologies. *Artificial Intelligence Review*, 22:85–126, 2004.

47. Alexander Ihler, Jon Hutchins, and Padhraic Smyth. Adaptive event detection with time-varying poisson processes. In *KDD '06: Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2006.

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

49. Anil K. Jain and Richard C. Dubes. *Algorithms for Clustering Data*. Prentice Hall, 1998.

50. Kevin Jeffay and Steve Goddard. A theory of rate-based execution. In *Proceedings of the 20th IEEE Real-Time Systems Symposium*, pages 304–314, December 1999.

51. Murray E. Jennex. Modeling emergency response systems. In *Proceedings of the 40th Hawaii International Conference on System Sciences*, 2007.

52. Eamonn Keogh, Stefano Lonardi, and Bill Ýuan chiĆhui. Finding surprising patterns in a time series database in linear time and space. In *SIGKDD*, 2002.

53. Daniel Kifer, Shai Ben-David, and Johannes Gehrke. Detecting change in data streams. In *Proceedings of the 30th VLDB Conference*, 2004.

54. Edwin M. Knorr and Raymond T. Ng. Algorithms for mining distance-based outliers in large datasets. In *Proceedings of the VLDB Conference*, pages 392–403, 1998.

55. V. Krishnamurthy, M. Faloutsos, M Chrobak, L. Lao, J-H Cui, and A. G. Percus. Reducing large internet topologies for faster simulations. *Networking*, 3462:328–341, 2005.

56. Sang Hoon Lee, Pan-Un Kim, and Hawoong Jeong. Statistical properties of sampled networks. *Physical Review E*, 73:016102–1–016102–7, 2006.

57. David Liben-Nowell and Jon Kleinberg. The link prediction problem for social networks. In *Proceedings of the 12th International Conference on Information and Knowledge Management (CIKM)*, 2003.

58. Jessica Lin, Eamonn Keogh, Stefano Lonardi, and Bill Chiu. A symbolic representation of time series with implications for streaming algorithms. In *Eighth ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery*, June 2003.

59. C. L. Liu and James W. Layland. Scheduling algorithms for multiporgramming in a hard-real-time environment. *Journal of the ACM*, 20(1):40–61, 1973.

60. Gregory R. Madey, Albert-Lásló Barabási, Nitesh V. Chawla, Marta Gonzalez, David Hachen, Brett Lantz, Alec Pawling, Timothy Schoenharl, Gábor Szabó, Pu Wang, and Ping Yan. Enhanced situational awareness: Application of DDDAS concepts to emergency and disaster management. In *Proceedings of the International Conference on Computational Science*, 2007.

61. Gregory R. Madey, Gabor Szabó, and Albert-Làszló 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. Springer, 2006.

62. Markos Markou and Sameer Singh. Novelty detection: a review. part 1: statistical approaches. *Signal Process.*, 83(12):2481–2497, 2003.

63. Markos Markou and Sameer Singh. Novelty detection: a review. part 2: neural network based approaches. *Signal Process.*, 83(12):2499–2521, 2003.

64. Michael Mitzenmacher and Eli Upfal. *Probability and Computing: Randomized Algorithms and Probabilistic Analysis*. Cambridge University Press, 2005.

65. Aloysius Mok. *Fundamental Design Problems of Distributed Systems for the Hard Real-Time Environment*. PhD thesis, Massechusetts Institute of Technology, 1983.

66. Andrew Moore. Very fast EM-based mixture model clustering using multiresolution kd-trees. In M. Kearns and D. Cohn, editors, *Advances in Neural Information Processing Systems*, pages 543–549, 340 Pine Street, 6th Fl., San Francisco, CA 94104, April 1999. Morgan Kaufman.

67. Real-time traffic routing from the comfort of your car. `http://www.nsf.gov/news/news_summ.jsp?cntn_id=107972`, August 2006.

68. M. E. J. Newman. Clustering and preferential attachment in growing networks. *Physical Review Letters E*, 64(025102), 2001.

69. M. E. J. Newman. The structure of scientific collaboration networks. *Proceedings of the National Academy of Sciences*, 98(2):404–409, January 2001.

70. Mark Newman. The structure and function of complex networks. *SIAM Review*, 45(2):167–256, 2003.

71. Alec Pawling, Nitesh V. Chawla, and Greg Madey. Anomaly detection in a mobile communication network. To appear: *Computational and Mathematical Organization Theory*.

72. Alec Pawling, Nitesh V. Chawla, and Greg Madey. Anomaly detection in a mobile communication network. In *Proceedings of the North American Association for Computational Social and Organization Science*, 2006.

73. Dan Pelleg and Andrew Moore. Accelerating exact k-means algorithms with geometric reasoning. In *Proceedings of the 5th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 277–281. ACM Press, 1999.

74. L. Portnoy, E. Eskin, and S. Stolfo. Intrusion detection with unlabeled data using clustering. In *ACM Workshop on Data Mining Applied to Security*, 2001.

75. Sridhar Ramaswamy, Rajeev Rastogi, and Kyuseok Shim. Efficient algorithms for mining outliers from large data sets. In *Proceedings of the ACM SIGMOD Conference on Management of Data*, 2000.

76. Matthew J. Rattigan and David Jensen. The case for anomalous link detection. In *Proceedings of the 4th International Workshop on Multi-Relational Mining*, 2005.

77. Tim Schoenharl, Ryan Bravo, and Greg Madey. WIPER: Leveraging the cell phone network for emergency response. *International Journal of Intelligent Control and Systems*, 11(4):209–216, 2006.

78. Timothy Schoenharl, Greg Madey, Gábor Szabó, and Albert-László Barabási. WIPER: A multi-agent system for emergency response. In *Proceedings of the 3rd International ISCRAM Conference*, 2006.

79. David W. Scott. *Multivariate Density Estimation: Theory, Practice, and Visualization*. John Wiley & Son, 1992.

80. Steven L. Scott. Detecting network intrusion using a markov modulated nonhomogeneous poisson process. http://www-rcf.usc.edu/sls/mmnhpp.ps.gz.

81. Steven L. Scott and Padhariac Smyth. The markov modulated poisson process and markov poisson cascade with applications to web traffic modeling. *Bayesian Statistics*, 7, 2003.

82. Lui Sha, Tarek Abdelzaher, Karl-Erik Arzén, Anton Cervin, Theodore Baker, Alan Burns, Giorgio Buttazzo, Marco Caccamo, John Lehoczky, and Aloysius K. Mok. Real time scheduling theory: A historical perspective. *Real-Time Systems*, 28:101–155, 2004.

83. Simone Sillem and Erik Wiersma. Comparing cell broadcast and text messaging for citizen warning. In *Proceedings of the 3rd International ISCRAM Conference*, 2006.

84. Carolyne Smart and Ilan Vertinsky. Designs for crisis decision units. *Administrative Science Quarterly*, 22:640–657, 1977.

85. Michael P. H. Stumpf, Carsten Wiuf, and Robert M. May. Subnets of scale-free networks are not scale-free: Sampling properties of networks. *Proceedings of the National Academy of Sciences*, 102(12):4221–4224, March 2005.

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

87. D. M. Titterington, A. F. M. Smith, and U. E. Makov. *Statistical Analysis of Finite Mixture Distributions*. John Wiley & Sons, 1985.

88. Alexey Tsymbal. The problem of concept drift: Definitions and related work. Technical Report TCD-CS-2004-15, Trinity College Dublin, 2004.

89. Jeffery S. Vitter. Random sampling with a reservoir. *ACM Transactions on Mathematical Software*, 11(1):37–57, 1985.

90. Duncan J. Watts and Steven H. Strogatz. Collective dynamics of ‘small-world’ networks. *Nature*, 393:440–442, June 1998.

91. Li Wei, Nitin Kumar, Venkata Lolla, Eamonn Keogh, Stefano Lonardi, and Chotirat Ann Ratanamahatana. Assumption-free anomaly detection in time series. In *Proceedings of the 17th International Scientific and Statistical Database Management Conference*, 2005.

92. Ian H. Witten and Eibe Frank. *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufman, 2nd edition, 2005.

93. Mark Wood. Cell@lert, for government-to-citizen mass communications in emergencies; it's about time. In *Proceedings of the 2nd International ISCRAM Conference*, 2005.

94. Jin Xu. *Mining and Modeling the Open Source Software Community*. PhD thesis, University of Notre Dame, 2007.

95. Dit-Yan Yeung and Calvin Chow. Parzen-window network intrusion detectors. In *Proceedings of the International Conference on Pattern Recognition*, 2002.

96. Dit-Yan Yeung and Yuxin Ding. Host-based intrusion detection using dynamic and static behavioral models. *Pattern Recognition*, 36:229–243, 2003.

This document was prepared & typeset with LaTeX 2_ε, and formatted with NDdiss2_ε *classfile (v3.0[2005/07/27]) provided by Sameer Vijay.*