APPROVAL SHEET

Title of Thesis: Efficient Local Algorithms for Distributed Data Mining in Large Scale Peer to Peer Environments: A Deterministic Approach

Name of Candidate: Kanishka Bhaduri
Doctor of Philosophy, 2008

Thesis and Abstract Approved: _____

Dr. Hillol Kargupta
Associate Professor
Department of Computer Science and
Electrical Engineering

Date Approved: _____

Curriculum Vitae

Name: Kanishka Bhaduri.

Permanent Address: 211-F Atholgate Lane, Baltimore MD-21229.

Degree and date to be conferred: Doctor of Philosophy, 2008.

Date of Birth: April 24, 1980.

Place of Birth: Kolkata, India.

Secondary Education: South Point High School, Kolkata, India, 1997.

Collegiate institutions attended:

- University of Maryland Baltimore County, Maryland, USA,
  Doctor of Philosophy, 2008.
- University of Miami, USA,
  2003–2004.
- Jadavpur University, Kolkata, India,
  Bachelor of Engineering, Computer Science and Engineering, 2003.

Major: Computer Science.

Professional publications:
Refereed Journals

1. **K. Bhaduri**, H. Kargupta. Distributed Multivariate Regression in Peer-to-Peer Networks. Statistical Analysis and Data Mining (SAM) Special Issue on Best of SDM'08. (submitted) 2008.
2. **K. Bhaduri**, R. Wolff, C. Giannella, H. Kargupta. Decision Tree Induction in Peer-to-Peer Systems. Statistical Analysis and Data Mining (SAM) accepted for publication. 2008.
3. K. Das, **K. Bhaduri**, K. Liu, H. Kargupta. Distributed Identification of Top-l Inner Product Elements and its Application in a Peer-to-Peer Network. IEEE Transactions on Knowledge and Data Engineering. Volume 20, Issue 4, pp. 475-488. April 2008.
4. K. Liu, **K. Bhaduri**, K. Das, P. Nguyen, H. Kargupta. Client-side Web Mining for Community Formation in Peer-to-Peer Environments. SIGKDD Explorations. Volume 8, Issue 2, pp. 11-20. December 2006.

5. R. Wolff, **K. Bhaduri**, H. Kargupta. A Generic Local Algorithm with Applications for Data Mining in Large Distributed Systems. IEEE Transactions on Knowledge and Data Engineering (TKDE) (submitted). 2007.

Book Chapter

1. **K. Bhaduri**, K. Das, K. SivaKumar, H. Kargupta, R. Wolff, R. Chen. Algorithms for Distributed Data Stream Mining. A chapter in Data Streams: Models and Algorithms, Charu Aggarwal (Editor), Springer. pp. 309-332. 2006.

Refereed Conference Proceedings

1. **K. Bhaduri**, H. Kargupta. An efficient local Algorithm for Distributed Multivariate Regression in Peer-to-Peer Networks. Accepted for publication at the 2008 SIAM International Data Mining Conference. (**Best of SDM'08**)
2. R. Wolff, **K. Bhaduri**, H. Kargupta. Local L2 Thresholding Based Data Mining in Peer-to-Peer Systems. SIAM International Conference in Data Mining, Bethesda, Maryland, USA. pp. 430-441. 2006.

Refereed Workshop Proceedings

1. K. Liu, **K Bhaduri**, K. Das, P. Nguyen, H. Kargupta. Client-side Web Mining for Community Formation in Peer-to-Peer Environments. SIGKDD workshop on web usage and analysis (WebKDD). Philadelphia, Pennsylvania, USA. 2006. (**Selected as the most interesting paper from the WebKDD workshop**)
2. **K. Bhaduri**, K. Das, H. Kargupta. Peer-to-Peer Data Mining. Autonomous Intelligent Systems: Agents and Data Mining. V. Gorodetsky, C. Zhang, V. Skormin, L. Cao (Editors), LNAI 4476, Springer. pp. 1-10. 2007.

Invited

1. S. Datta, **K. Bhaduri**, C. Giannella, R. Wolff, H. Kargupta. Distributed Data Mining in Peer-to-Peer Networks. IEEE Internet Computing special issue on Distributed Data Mining. Volume 10, Number 4, pp. 18-26. 2006.

Professional positions held:

- Research Assistant. (05/2004 – 03/2008).
  Distributed Adaptive Discovery and Computation Lab, Department of Computer Science and Electrical Engineering, University of Maryland Baltimore County (UMBC).

- Software Internship. (05/2007 – 08/2007).
  Symantec Corporation, Columbia, Maryland.
- Teaching Assistant. (08/2003 – 05/2004).
  Department of Computer Science, University of Miami, Florida.

ABSTRACT

Title of Dissertation: Efficient Local Algorithms for Distributed Data Mining in Large Scale Peer to Peer Environments: A Deterministic Approach

Kanishka Bhaduri, Doctor of Philosophy, 2008

Thesis directed by: Dr. Hillol Kargupta
Associate Professor
Department of Computer Science and
Electrical Engineering

Peer-to-peer (P2P) systems such as Gnutella, Napster, e-Mule, Kazaa, and Freenet are increasingly becoming popular for many applications that go beyond downloading music files without paying for it. Examples include P2P systems for network storage, web caching, searching and indexing of relevant documents and distributed network-threat analysis. These environments are rich in data and this data, if mined, can provide valuable source of information. Mining the web cache of users, for example, may often give information about their browsing patterns leading to efficient searching, resource utilization, query routing and more. However, most of the off-the-shelf data analysis techniques are designed for centralized applications where the entire data is stored in a single location. These techniques do not work in a highly decentralized, distributed environment such as a P2P network. We need distributed data mining algorithms that are fundamentally *local*, scalable, decentralized, asynchronous and anytime to solve this problem.

This research proposes *DeFraLC*: a **De**terminsitic **Fra**mework for **L**ocal **C**omputation of functions defined on data distributed in large scale (peer to peer) systems. Computing global data models in such environments can be very expensive. Moving all or some of the data to a central location does not work because of the high cost involved in centralization. The cost increases even more under a dynamic scenario where the peers' data and the network topology change arbitrarily. In this dissertation we have focused on developing

algorithms for deterministic function-computation in large scale P2P environments. Our algorithmic framework is **local** which means that a peer can compute a function based on the information of only a handful of nearby neighbors and the communication overhead of the algorithm is upper bounded by some constant, independent of the size of the system. As a consequence, several messages can be pruned, leading to excellent scalability of our algorithms.

The first algorithm that we have developed — *PeGMA*, **Pe**er-to-Peer **G**eneric **M**onitoring **A**lgorithm — is capable of computing complex functions defined on the average of the horizontally distributed data. This generic algorithm is extremely accurate, highly scalable and can seamlessly adapt to changes in the data or the network. Following *PeGMA*, several interesting algorithms can be developed such as the L2 norm monitoring of distributed data which is a very powerful primitive. Using a two step feedback loop, a number of data mining algorithms have been proposed. The first step uses the local algorithm to raise a flag whenever the current data does not fit the function. The second step uses a feedback loop to sample data from the network and build a new function. The correctness of the local algorithm guarantees that once the computation terminates each peer has the same result compared to a centralized scenario. We propose solutions for P2P k-means monitoring, eigen monitoring and multivariate regression in P2P environments. Furthermore, we have shown how a complex data mining algorithm such as decision tree induction can be developed for P2P environments. Finally we have implemented all of the algorithms in a Distributed Data Mining Toolkit (DDMT) [44] developed at the DIADIC research lab at UMBC. Our extensive experimental results show that the proposed algorithms are accurate, efficient and highly scalable.

EFFICIENT LOCAL ALGORITHMS FOR DISTRIBUTED DATA MINING IN LARGE SCALE PEER TO PEER ENVIRONMENTS: A DETERMINISTIC APPROACH

by

Kanishka Bhaduri

Dissertation submitted to the Faculty of the Graduate School
of the University of Maryland in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2008

Dedicated to Dadon...who was always proud of my achievements

ACKNOWLEDGMENTS

Finally it is time for me to acknowledge all those who inspired me, supported me and helped me to get to the place where I am today.

Firstly, I would like to thank my advisor Dr. Hillol Kargupta for providing me the support and motivation to do my job well. His advice will always be valuable to me. I also want to thank Dr. Ran Wolff and Dr. Chris Giannella for introducing me to my research area and guiding me through it. Without them, I would not have accomplished this today. They were always there in times of need with their valuable suggestions. Before moving on, I would also like to thank the UMBC CSEE Department for giving me this opportunity to fulfil my dream; I have a lot of happy memories associated with this place.

Nothing I say about my parents is going to be enough to describe their love and support for me throughout my life. I feel, without my mother I would not have been the person I am today. Two people who have been my pillars of support throughout my graduate life are Appa and Aunty. Their constant words of support and inspiration helped me breeze through trying times and part of my success definitely belongs to them. I also have to acknowledge Amy who knowingly and unknowingly provided me with the much needed humor in stressful times in all these years. Another person who deserves a special mention here is my first mentor Dr. Suman Chakraborty. He introduced me to the world of research when I had barely started to understand computer science. I have always considered him to be my role model.

Finally, I want to thank my wife, best friend and colleague Sherin. This thesis is as much hers as it is mine. She has been the singlemost source of inspiration in my life and without her I would never have thought of pursuing my PhD. I want to thank her for all the intellectual discussions, for all her constructive criticisms, her love and unconditional support that helped me to be the person I am today, both personally and professionally. I

hope she finishes her PhD soon so that we can be together again.

I also want to mention my labmates Kun Liu, Haimonti Dutta, Souptik Datta and Sourav Mukherjee — working with all of them has been a pleasure. My graduate life at UMBC paved the path for some very important friendships in my life. I will always remember Kishalay, Nicolle, Aarti and Aseem for all their help and for all the fun things we did together.

Finally, I would like to thank all my committee members for providing me with their valuable feedback on my thesis and for working with me under severe time constraints. Overall, it has been a rewarding experience that I will cherish for a long time to come.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

List of Algorithms

Chapter 1

INTRODUCTION

1.1 Introduction

Proliferation of communication technologies and reduction in storage costs over the past decade have led to the emergence of several distributed systems. These environments are rich in data; however unlike traditional systems where the data and computational resources are at one central location, these systems are distributed both in terms of data and resources. As a result, mining in such environments naturally calls for proper utilization of these distributed resources. Moreover, in many privacy sensitive applications different, possibly multi-party, data sets collected at different sites must be processed in a distributed fashion without collecting everything to a single central site. However, most off-the-shelf data mining systems are designed to work as a monolithic centralized application. They normally down-load the relevant data to a centralized location and then perform the data mining operations. This centralized approach does not work well in many of the emerging distributed, ubiquitous, possibly privacy-sensitive data mining applications due to computation, communication, and storage constraints.

Distributed Data Mining (DDM) offers an alternate approach to address this problem of mining data using distributed resources. DDM pays careful attention to the distributed resources of data, computing, communication, and human factors in order to use them in a near optimal fashion. Distributed data mining is gaining increasing attention in today's

world for advanced data driven applications.

Peer-to-peer (P2P) systems are emerging as a choice of solution for a new breed of applications such as file sharing, collaborative movie and song scoring, electronic commerce, Internet chat, webcast surveillance using sensor networks to name a few. P2P systems can be viewed as an extreme case of DDM applications — they are huge (sizes are of the order of tens of thousands or millions in many cases), there is no coordinator node, the connections among the nodes are ad hoc, the node and link failures are unpredictable, data changes with time (streaming scenario) just to mention a few. Traditional DDM algorithms cannot often be deployed in such environments mainly because of scalability issues, asynchronous nature of such environments, streaming data — to mention a few.

This dissertation proposes a framework — *DeFraLC* (**De**terministic **F**ramework for **L**ocal **C**omputing) — for distributed data mining in large P2P systems. The algorithmic framework proposed is deterministic and provably correct in the sense that the result produced by our framework is the same that would be produced if all the data were accumulated at a central site and then an off-the-shelf centralized data mining algorithm was run. Since these systems typically consist of millions of nodes or peers and the system is constantly changing, it is very important to compute the results efficiently. Local computation, as we define in this dissertation, guarantees an upper bound on the communication complexity at each node (independent of the network size), thereby providing high scalability of these algorithms. Each node contacts only a few of the other nodes in the network to generate a global mode. The *DeFraLC* framework handles node and data changes with ease; the model adapts automatically whenever the system changes. We show the generic nature of *DeFraLC* by proposing a number of algorithms which are direct application of *DeFraLC*.

1.2 Motivation

Data mining has been defined as “the nontrivial extraction of implicit, previously unknown, and potentially useful information from data” [60] and “the science of extracting useful information from large data sets or databases” [73]. It involves searching through large amounts of data and picking out relevant information.

P2P networks are quickly emerging as huge information systems. Through networks such as Kazaa, e-Mule, BitTorrents and more consumers can share vast amounts of data. While initial consumer interest in P2P networks was focused on the data itself, more recent research such as P2P web community formation argues that the consumers will greatly benefit from the knowledge locked in the data [115] [37].

For instance, music recommendations and sharing systems are a thriving industry today [3][152] -- a sure sign of the value consumers have put on this application. However, all existing systems require that users submit their listening habits, either explicitly or implicitly, to centralized processing. Such centralized processing can be problematic because it can result in severe performance bottleneck. Several distributed association rule mining algorithms (e.g. the one by Wolff *et al.* [196]) have shown that centralized processing may not be a necessity by describing algorithms which compute association rules (and hence, recommendations) in-network; processing the data in-network means that it is very efficient and scalable. Moreover Mierswa *et al.* [123] have demonstrated the collaborative use of features for organizing mucic collections in a P2P setting.

Another application which offers high value to the consumers is failure determination [145][206]. In failure determination, computer-log data which may have relation to the failure of software and this data is later analyzed in effort to determine the reason for the failure. Data collection systems are today integral to both the Windows and Linux operating systems. Analysis is performed off-line on a central site and often uses knowledge discovery methods. Still, home users often choose not to cooperate with current data col-

lection systems because they fear for privacy and currently there is no immediate benefit to the user for participating in the system. Collaborative data mining for failure determination can be very useful in such scenarios. Such a privacy preserving P2P misconfiguration diagnosis technique has been proposed by Huang et al [82].

Consider a P2P system such as Napster [134] or Gnutella [68]. In most of these large scale systems, users download music files or other files based on certain preferences. There are several interesting questions that can be asked in such environments. For example, does the type of music downloaded change over time ? Or, is there a particular type of file or genre of music preferred by users at a certain time of the year (*e.g.* during the Christmas) ? Can users be clustered based on their preferences ? Answers to many such questions can be crucial from the system's point of view — the resources necessary can change over time and it makes sense to reinvest the unused resources in a better way.

However, it is not always possible to centralize the data and build models of it. The bandwidth required to centralize the data may be too expensive. The storage required at the central repository may also be unacceptable. Above all, if the data is non-stationary, it may so happen that the rate at which the data changes is faster than the rate at which data can be propagated through the network, making distributed monitoring algorithms the only alternative. Moreover traditional client server based DDM algorithms do not scale to millions of peers and hence they also fail to address the issues in P2P applications. This motivates us to develop the *DeFraLC* framework.

1.3 Problem Statement

This dissertation addresses the following problem. Consider a large P2P network consisting of millions of nodes or peers and connected via an underlying communication infrastructure. Each node has some data which is known only to itself. We call this the local data of the node. Each node can exchange messages with its neighbors. The system is con-

stantly changing both with respect to the nodes and the data contained at each node. This research aims at answering the question: "How can data mining techniques or algorithms be developed or applied that can extract useful knowledge from the union of all data of all peers in these networks under the following constraints (1) low communication overhead, (2) no synchronization, (3) failure resistance in case of nodes departing or joining, and (4) quality of results comparable to centralized mining techniques."

1.4 Contributions

In this dissertation we have systematically studied the area of deterministic local algorithms for data mining in P2P environments. A generic framework — *DeFraLC* (**De**terministic **F**ramework for **L**ocal **C**omputing) — has been proposed for developing many deterministic local algorithms. The strength yet simplicity of the framework is demonstrated by developing a number of algorithms based on it — the L2 Thresholding, Means monitoring, k-means monitoring, eigen vector monitoring, multivariate regression and decision tree induction.

The specific contributions of this dissertation are highlighted next.

1. We have identified a common principle which can be used to develop extremely complex efficient exact (deterministic) local algorithms, proving a main theorem from which a large number of deterministic local algorithms can be developed.

2. Based on the above theorem, we have developed the *DeFraLC* framework. The **Pe**er-to-Peer **Ge**neric **M**onitoring **A**lgorithm (*PeGMA*) proposed in *DeFraLC* can be used to compute arbitrarily complex functions of the average of the data in a distributed system. It can also be extended to compute functions on other linear combinations of data, including weighted averages of selections from the data.

3. Third, the *DeFraLC* can be used for monitoring, and consequent reactive updating

of any model of horizontally distributed data. Consequently, we have shown how the *DeFraLC* framework can be applied to track the average of distributed data, the first eigenvector of that data, the k-means clustering of that data and multivariate regression of the data.

4. Finally, we have shown how a complex data mining algorithm such as decision tree induction can be developed for P2P systems which is robust to data and network changes, suffers low communication overhead, offers high scalability and is asynchronous. In the process we have demonstrated how complex functions such as gini or entropy information gain need to be replaced by simpler ones such as misclassification gain to aid in the tree building process.

5. Lastly, we have implemented all of the proposed algorithms in a Distributed Data Mining Toolkit (DDMT) [44], developed by the DIADIC Research Lab at UMBC.

1.5 Dissertation Organization

This dissertation is organized as follows.

Chapter 1: This chapter introduces the domain of research, motivates the problem, presents the problem statement and contributions, and states the organization of the dissertation.

Chapter 2: This chapter presents a background study of the different types of P2P networks and the characteristics of the P2P data mining algorithms. Then it discusses the existing algorithms and techniques for distributed data mining, P2P data mining, distributed data stream mining, sensor networks, GRID systems, P2P information retrieval and multiagent systems. Finally this chapter discusses some applications of the P2P networks.

Chapter 3: This chapter presents the *DeFraLC* framework (**De**terministic **F**ramework for **L**ocal **C**omputing). The **Pe**er-to-Peer **Ge**neric **M**onitoring **A**lgorithm (*PeGMA*), the main

component of *DeFraLC* is capable of computing complex functions defined on the average of the global data. Further, this chapter shows how the generic algorithm can be used for L2 norm thresholding, mean monitoring, k-means monitoring, eigenvector monitoring and multivariate regression in large P2P settings.

Chapter 4: This chapter presents an efficient P2P decision tree induction algorithm which is asynchronous and robust to data and network changes. It shows (1) how the algorithm is efficient compared to brute force centralization of data statistics and (2) the quality is comparable to such a centralized algorithm.

Chapter 5: This chapter concludes the dissertation and presents some prospective areas of future research.

Chapter 2

BACKGROUND AND RELATED WORK

2.1 Introduction

P2P systems such as Gnutella, BitTorrents, e-Mule, Kazaa, and Freenet are increasingly becoming popular for many applications that go beyond downloading music without paying for it. Examples include P2P systems for network storage, web caching, searching and indexing of relevant documents and distributed network-threat analysis. The next generation of advanced P2P applications for bioinformatics[1], client-side web mining [115] and large-scale web search[2] are likely to need support for advanced data analysis and mining. This has resulted in the development of several distributed data mining algorithms specifically focused towards knowledge extraction from such large asynchronous networks.

Our goal in this chapter is to first familiarize the reader with the different types of P2P networks. We then follow the discussion with the characteristics of the algorithms suitable for P2P networks. Finally we present some related work in the area of this research. For convenience, we have categorized the related work into the following areas. We start with the related work in DDM in Section 2.5 followed by the related work in P2P computing in Section 2.6. Since data stream mining in distributed environments has seen many algorithms similar to P2P data mining, we discuss them in Section 2.7. Sensors are often

[1] http://smweb.bcgsc.bc.ca/chinook/index.html
[2] http://www.yacy.net/

deployed both in military and civilian applications and they form an ad-hoc network. Although currently their communication and network infrastructure are very structured and hierarchical, the next generation of sensor net applications may have more decentralized control such as in typical P2P networks. Because of its proximity to this area of research, we summarize the related work in data mining in sensor networks in Section 2.8. We discuss the related work in P2P information retrieval in Section 2.9 followed by the related work in data mining in GRIDS in Section 2.10. Finally, we present some work in the area of multi-agent systems and distributed AI in Section 2.11. Before we end this chapter, we also describe some exciting application areas of the P2P paradigm of data mining and computing.

2.2 Client-Server Networks and Peer-to-Peer Networks

Advances in computing and communication over wired and wireless networks have resulted in many pervasive distributed computing environments for example the Internet, Intranets, local area networks, ad hoc wireless networks, sensor networks and peer-to-peer networks. Most of the initial effort was directed towards developing systems based on client-server architecture such as most modern web servers, file servers, mail servers, print servers, e-commerce applications and more. Figure 2.1 shows a system based on client server architecture. In the figure the clients try to connect to a file server, application server, web server or database server. This architecture became popular mainly because of the simple synchronous nature of the communication between the server and the client. Since the data is stored mainly at the server, it is relatively easy to ensure security and privacy. However as more and more of these systems evolved, several shortcomings were realized. With increased clients and more requests from each client, traffic congestion between the server and the client became heavy, leading to reduced throughput. Moreover in many cases several clients sat idle, leading to unbalanced load distribution. Another

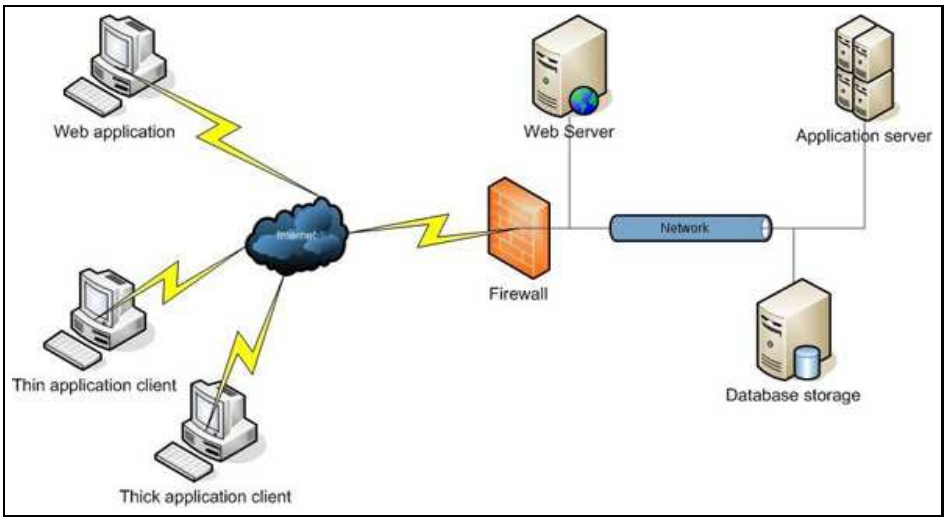


FIG. 2.1. A client server architecture showing three clients and three servers — a web server, application server and database server. Image source http://www.sqapartners.com/images/webmatrix2_590.jpg.

major drawback was the lack of robustness to failed clients. Since in a typical client server-based distributed system the data is not replicated but rather kept at a single site, a node failure is devastating for the entire system.

To alleviate some of these issues, Peer-to-Peer networks were developed. The term "Peer-to-Peer" (P2P) refers to a class of systems and applications that employ distributed resources to perform critical functions in a decentralized manner [124]. P2P networks are typically used for connecting nodes via largely ad hoc connections and hence all the nodes are equal in functionality. Each node acts as both the *serv*er and cli*ent* and hence peers are often referred to as *servent*. With the pervasive deployment of computers, P2P technology is increasingly receiving attention in research, product development, and investment circles. Sharing static content files such as audio, video, data or anything in digital format is very common using P2P technology. Realtime data such as telephony traffic, streaming audio

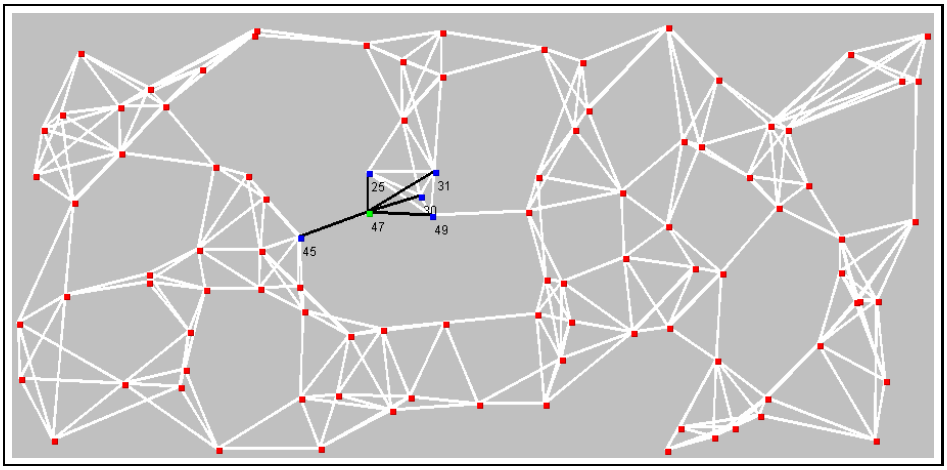


FIG. 2.2. A P2P network of 100 nodes showing the ad-hoc structure.

and tv channels[3] is handled in a P2P manner. There are several advantages of a P2P network over the client-server model. Due to the absence of any centralized node, there is no single point of failure. Moreover, all clients provide resources, including bandwidth, storage space, and computing power. Thus, as nodes arrive and demand on the system increases, the total capacity of the system also increases (because all nodes act as both the server and client). This is not true of a client-server architecture with a fixed set of servers, in which adding more clients could mean slower data transfer for all users. Since this research aims at developing algorithms for data mining in P2P networks, the rest of this document will focus on this model of computing.

Depending on how the individual peers are connected, there are different types of P2P networks — we present an overview of the different types in the next section.

[3]http://www.sopcast.org/

2.3 Types of Peer-to-Peer Networks

Several classifications exist for P2P networks. In this section we follow the one discussed by Lv *et al.* [117]. The authors classify the different architectures of P2P networks into three classes — (1) Centralized, (2) Decentralized but structured, and (3) Decentralized and unstructured.

2.3.1 Centralized

In this type of P2P networks, although there are several nodes or peers, the point of control or communication is through a central server. In early days of its advent, Napster [134] followed this type of P2P topology. All queries were handled through the central node and hence this node needed to have an index of all the files in the network in order to provide an answer to the query. Naturally, the most serious disadvantage to this structure is a single point of failure. Also these structures are not scalable due to the dependence on the central node.

2.3.2 Decentralized but structured

In order to circumvent some of the problems associated with the earlier architecture, designers came up with a decentralized network structure. In this model of architecture, there is no concept of central node. However, in many cases, there is significant structure imposed on the network. For example, in many applications, there is an implicit assumption of an overlay tree structure in such networks [195]. There are more examples of such networks as provided in CHORD [175], CAN [158], PASTRY [160] and more. In these networks, the major idea is to use a Distributed Hash Table (DHT) to aid in the searching of contents in such networks. The files to be searched are not placed in random nodes; rather they are hashed to a number and the files are placed at the nodes corresponding to that number (node id).

2.3.3 Decentralized and unstructured

These are systems in which there is neither a centralized directory nor any precise control over the network topology or file placement. Gnutella [68] is an example of such designs. The resultant topology has certain properties, but the placement of files is not based on any knowledge of the topology. To find a file, a node queries its neighbors. The most typical query method is flooding, where the query is propagated to all neighbors within a certain radius. These unstructured designs are extremely resilient to nodes entering and leaving the system. These networks are easy to maintain as well.

With such varied types of P2P networks, it is easy to see that not all algorithms are suitable for all the P2P architectures. In the next section we present some desired characteristics of P2P data mining algorithms.

2.4 Characteristics of Peer-to-Peer Data Mining Algorithms

As pointed out in the last section, the computational environment of P2P systems is quite different from the ones for which traditional data mining algorithms are intended. As such, these environments demand a new breed of algorithms for efficient operation. Below are a list of operational characteristics desired for data mining algorithms developed for such networks.

1. *Communication Efficient* — P2P data mining algorithms must be able to work in a communication efficient manner. Although, many P2P systems are designed for sharing large data files (e.g. music, movies), a distributed data mining system that involves analyzing such data, may not have the luxury to frequently exchange large volume of data among the nodes in the P2P network simply for data analysis. Ideally, it should not move any data (or at the maximum share minimal data), but only exchange ‘knowledge’ about data for mining purpose. Data mining in a P2P environ-

ment should be “light-weight”; it should be able to perform distributed data analysis with minimal communication overhead.

2. *Scalability* — One of the most desired characteristics of any P2P algorithm is scalability. Since modern P2P systems can grow up to millions of peers (*Skype* presently has around 50 million peers), the computational and communication (bandwidth) resource requirement for such algorithms should be independent of the size of the system. Communication efficient algorithms are, in general, expected to be scalable. *Local algorithms*, which we define in the next chapter offers a new way to develop highly scalable P2P algorithms.

3. *Anytimeness* — In many real world P2P systems *viz.* sensor networks, data at the nodes often changes with time (streaming scenario). Any algorithm that needs to begin computation from scratch whenever the data changes is thus inappropriate for our goals since it would (1) take too many resources for every data change, and (2) take a long time to give the result. Typically, in a streaming scenario we want answers to the problems as and when required — incremental algorithms are can provide with the solutions quickly. Furthermore, since the rate of data-change may be higher than the rate of computation in some applications, the algorithm should be able to report a partial, ad hoc solution, at any time.

4. *Asynchronism* — Due to the massive scale of P2P systems and its volatile nature, P2P algorithms are desired to be asynchronous. Synchronized algorithms are likely to fail due to connection latency, limited bandwidth or node failures.

5. *Decentralization* — Although some P2P systems still use central servers for various purposes, ideally any algorithm designed for peer-to-peer systems should be able to run in absence of any coordinator (server or router) and calculate the result in-network rather than collect data in a single peer. This immediately points out the

advantages — (1) there is no traffic bottleneck at the central node, and (2) there is no single point of failure.

6. *Fault-tolerance* — In P2P systems, it is not unusual for several peers to leave and join the system at any given moment. Thus,the system should be able to recover from the failure of peers, and the subsequent loss of data. In other words, the algorithm must be 'robust' to network and data changes.

7. *Privacy and Security* — Since P2P networks consists of heterogenous entities working in a collaborative fashion, privacy is an important issue. Similar to any other large distributed system, security is also an important issue in P2P data mining system.

In many cases, developing algorithms that satisfy all of the above requirements is quite difficult. In this dissertation, therefore, we will focus on certain characteristics of the P2P algorithms.

In the next few sections we present some literature related to our area of research. Since our research relates to many different areas of study, we have divided this section into the following subsections. The first section (Section 2.5) discusses the related work in distributed data mining. Section 2.6 discusses the related work in P2P data mining. The next section (Section 2.7) presents the related work in distributed data stream mining followed by the related work in data mining in sensor networks in Section 2.8. Section 2.10 offers some details in the area of data mining in Grids. Finally, we present some work in the area of multi-agent systems and distributed AI in Section 2.11.

2.5 Related work: Distributed Data Mining

In this section we present a brief overview of the existing work on Distributed Data Mining or (DDM) in general. DDM, as the name suggests, deals with the problem of data analysis in environments with distributed data, computing nodes, and users. This area has

seen considerable amount of research during the last decade. For an introduction to the area, interested readers are referred to the books by Kargupta *et al.* [93][94]. A related of research is the area of system development and algorithm development for distributed systems. The book by Ghosh [65] presents a detailed study of the different types of algorithms developed for distributed systems.

Depending on how the data is distributed across the various sites, a natural way to categorize the DDM algorithms is as follows:

- *Horizontally partitioned* — in which all the features are present in all the sites, but the samples/data tuples are distributed across the sites.
- *Vertically partitioned* — in which the features are distributed across all the sites (may be overlapping or disjoint).

DDM algorithms have been proposed for many popular data analysis tasks. In the next two subsections, we present a brief sampling from each major category. We focus primarily on distributed clustering and classification since this thesis proposed algorithms for doing the same in a P2P domain. Interested readers are referred to the books by Kargupta *et al.* [93][94], the distributed data mining bibliography maintained at UMBC [43] and several surveys [201][202] for more in-depth discussions.

2.5.1 Distributed Classification

Ensemble based classifier learning is well suited to distributed data mining framework. In ensemble techniques such as bagging [21], boosting [62] and random forests [22] several weak classifiers are induced from different partitions of the data and then they are combined using voting techniques or otherwise to produce the output. These techniques can be adopted for distributed learning by inducing a weak classifier from each distributed data site and then combining them at a central location. In the literature this is popularly

known as the meta-learning framework [31][30]. Several strategies for combining the classifiers have been proposed such as voting, arbitration and combiner. The meta learning framework for homogenous dataset is implemented as part of the JAM system [176].

The problem of learning from heterogeneously partitioned data has been addressed by several researchers. Park and his colleagues [148] have developed algorithms for learning from heterogenous datasets using an evolutionary technique. Their work first builds local classifiers and then identifies a selection of tuples that none of these local classifiers can correctly classify. These tuples are centralized and a new classifier is build on this tuples. The classification of a new tuple is based on either a collection of local classifiers or the centralized one.

Caragea *et al.* [29] presented a decision tree induction algorithm for both distributed homogenous and heterogenous environments. Noting that the crux of any decision tree algorithm is the use of an effective splitting criteria, the authors propose a method by which this criteria can be evaluated in a distributed fashion. More specifically the paper shows that by only centralizing summary statistics from each site e.g., counts of instances that satisfy specific constraints on the values of the attributes to one location, there can be huge savings in terms of communication when compared to brute force centralization. Moreover, the distributed decision tree induced is the same compared to a centralized scenario. Their system is available as part of the INDUS system.

A different approach was taken by Giannella *et al.* [66] and Olsen [141]. They used Gini information gain as the impurity measure and showed that Gini between two attributes can be formulated as a dot product between two binary vectors. To cut down the communication complexity, the authors evaluated the dot product after projecting the vectors in a random subspace. Instead of sending either the raw data or the large binary vectors, the distributed sites communicate only these projected low-dimensional vectors. The paper shows that using only 20% of the communication cost necessary to centralize the data,

they can build trees which are at least 80% accurate compared to the trees produced by centralization.

An order statistics based approach to combining classifier ensembles generated from heterogenous data has been proposed by Tumer and Ghosh [189].

The collective data mining framework (CDM) by Kargupta *et al.* [92] proposes algorithms for data mining from heterogenous data sites using orthogonal basis functions. The basic idea is to represent the model to be built using an orthonormal basis set such as fourier coefficients. These coefficients are then evaluated at each local site and they are transferred to a central location. Coefficients involving cross terms between the sites are evaluated at the central site after centralizing a sample of the data. Several data mining algorithms have been proposed using this technique. Bayes net from distributed heterogenous data [33], decision trees [96], distributed multivariate regression [78], distributed principle component (PCA) and data clustering [95] are some of the algorithms developed using the CDM framework.

2.5.2 Distributed Clustering

Several techniques have been proposed in the literature for distributed clustering of data.

Kargupta *et al.* [95] have developed a principle component analysis (PCA) based clustering technique on the CDM framework for heterogeneously distributed data. Each local site performs PCA, projects the local data along the principle components, and applies a known clustering algorithm. The communication complexity of such a technique is much smaller than centralizing all the data.

Klusch *et al.* [102] considered the problem of distributed clustering over homogenously distributed data using kernel-density. They use a local definition of density based cluster and points which have same density (according to some kernel function) are put in

the same cluster. They build local clusters of the data and transmit these to a central site. The central site then combines these clusters.

k-means clustering of data is a popular data mining technique. Distributed versions of k-means clustering algorithms have been proposed by various researchers till date. Eisenhardt *et al.* [53] proposed a distributed method for document clustering using k-means. Their technique is enhanced with a "probe and echo" mechanism for updating cluster centroids. The algorithm is synchronized and each round corresponds to a k-means iteration. There is one designated initiator node. It launches a probe message to all its neighbors and sends its local centroids and weights. Whenever a node P_j receives a probe message from its neighbor P_i, it first updates its current centroids with the ones received from P_i and then forwards the probe to all its neighbors (except P_i). When a site has received a probe or each from everyone, it forwards the message to the one from which it first received the probe. This process continues until a termination criteria stops the iterations.

Based on this technique, several extensions have been proposed. Banyopadhyay *et al.* [7] have proposed an algorithm for k-means clustering based on sampling technique. To relax the synchronization assumptions, Datta et al. [40] later proposed an asynchronous version of k-means algorithm suitable for P2P networks. Dhillon and Modha [46] have developed a parallel implementation of the k-means algorithm on homogenously partitioned dataset. Similar technique for k-harmonic mean has been proposed by Forman and Zhang [58].

A different approach to distributed clustering is to generate local models and then combine them at a central site. These techniques are (1) asynchronous, (2) offer good communication complexity compared to centralization of data, and (3) offers privacy preserving solutions in many domains.

Johnson and Kargupta [87] have proposed a hierarchical clustering algorithm on heterogeneously distributed data. The idea is to generate local dendograms and then combine

them at a central site.

Lazarevic *et al.* considered the problem of combining spatial clusterings to produce a global regression-based classifier on a homogenously distributed data.

Several other techniques for distributed heterogenous data clustering have been proposed using ensemble approach such as [179] and [61].

2.6 Related work: Data Mining in Peer-to-Peer Environments

P2P data mining has very recently emerged as a subfield of DDM, specifically focusing on algorithms which satisfy the properties mentioned in Section 2.4. Due to its nascency, this area of research has little prior work. Datta *et al.* [39] present an overview of this topic.

In the next few sections we present an overview of the different types of P2P data mining algorithms. Table 2.1 shows the taxonomy of the different P2P data mining algorithms.

P2P Alg.	Approximate Alg.	Probabilistic Alg.	Sampling Alg.
			Gossip-based Alg.
		Deterministic Alg.	
	Exact Alg.	Flooding Alg.	
		Convergecast Alg.	
		Local Alg.	
		Cluster-based Alg.	

Table 2.1. Categorization of the different P2P data mining algorithms.

2.6.1 Approximate Algorithms

Approximate algorithms, as the name suggests, computes the approximate data mining results. The approximation can be probabilistic (Section 2.6.1) or deterministic (Section 2.6.1).

Probabilistic Approximate Algorithms: Probabilistic algorithms rely on the results of sampling theory from statistics to derive bounds on the quality of the results.

1. Sampling-based algorithms: In sampling-based algorithms peers sample data (using some variations of graph random walk as proposed by Datta *et al.* [41]) from their own partition and that of several neighbors' and then build a model assuming that this data is representative of that of the entire set of peers. Thus, the algorithms claim nothing on the accuracy of the resulting model in the general case. Examples for these algorithms include the P2P k-Means algorithm by Banyopadhyay *et al.* [7], the naive Bayes algorithm by Kowalczyk *et al.* [104], and more. Banyopadhyay *et al.* [7] talk about a distributed k-Means algorithm designed for a horizontally partitioned scenario, such that only the centroids of the local sites need to be communicated. The paper presents a theoretical proof of quality and convergence (in some restricted cases *e.g.* when the data is sampled uniformly from the network). This work was one of the first efforts of engineering a well-known data mining technique to work in a P2P domain. Datta *et al.* [40] later enhanced this algorithm to work in an asynchronous environment. However it still lacks theoretical proof of quality and convergence.

More recently, Das *et al.* have developed an algorithm [37] for identifying significant inner product entries in a P2P network in a horizontally partitioned data distribution scenario. Inner product is an important primitive in data mining, useful for many techniques such as distance computation, clustering, classification and more. The proposed algorithm uses a variant of Metropolis-Hastings random walk [121] to draw random samples from the network and using the results from the ordinal decision theory bounds the quality of the result and the communication complexity. Extending further, Liu *et al.* [115] proposed an algorithm for community formation in P2P environments using client-side browser history and inner product computation. The latter paper also talks about preserving the users' privacy using SMC protocols.

2. Gossip-based algorithms: The second category, gossip-based algorithms, relies on properties of random walks on graphs to provide estimates for various statistics of data stored in the graph. Gossip-based computation was first introduced by Kempe *et al.* [99], in which they showed that each peer, by contacting a small number of nodes chosen at random, can get the result of the computation exponentially fast. Boyd *et al.* [18] enhanced the protocol for general graphs. The most important quality of gossip-based algorithms is that they provide probabilistic guarantees for the accuracy of the result. However, the first gossip-based algorithms required that the algorithm be executed from scratch if the data changes in order to maintain those guarantees. This problem was later addressed by Mehyar *et al.* [119], and by Jelasity *et al.* [85]. Mehyar *et al.* [119] propose a graph Laplacian-based approach to compute the average of a set of points in a P2P network. Each peer has a number say x_i and an estimate of the global average z_i. The paper shows that the rate at which $z_i \rightarrow \frac{1}{n}\sum_{i=1}^{n} x_i$ is exponential, thereby achieving fast convergence.

Deterministic Approximate Algorithms: The probabilistic techniques, as discussed in the previous sections use randomized algorithms and hence their output depends on the input and some other random variables. As such, these algorithms can only guarantee the quality of the results on average and the variance of the process is also bounded. A single trial can yield a very different answer. To counter these problems, researchers have proposed a new genre of approximation technique which we call here the deterministic approximation technique. In such techniques, the output is only dependent on the input. The results, although approximate, hold true for every single trial. This makes them very attractive for many real-life problems including data mining in distributed environments.

A popular method of performing deterministic approximation technique is to use the method of variational approximation as proposed by Jordan *et al.* [89] and by Jaakkola [83]. The variational approximation technique consists of three steps:

1. Pose the original problem as a optimization problem. Define an objective function

for the same such that the optimal solution of this objective function gives the exact solution of the original problem.

2. Develop a search strategy to search in this transformed space.

3. Since for many problems this search can be intractable, apply approximations that make this search computationally feasible. Note that this step makes (1) the variational technique feasible and (2) an approximate technique rather than an exact technique.

Several approaches exist in the literature based on the variational approximation technique. Jordan and Jaakkola have proposed solutions for problems such as inferencing, parameter estimation using EM algorithm, gaussian mixture models and more. Bayesian parameter learning using variational technique have been proposed by Murphy [132]. While most of these techniques work for centralized scenarios, Mukherjee [130] has proposed distributed algorithms for variational approximation. The distributed inferencing or VIDE framework [129] aims to solve the problem of target tracking in wireless sensor networks. Further, he proposed solutions for distributed linear regression using the same variational framework [131].

2.6.2 Exact Algorithms

Exact algorithms form an exciting paradigm of computation whereby the result generated by the distributed algorithms is exactly the same if all the peers were given all the data. Thus, contrary to approximate techniques, these algorithms produce the correct result everytime they are executed. Depending on how these algorithms compute the correct result, exact algorithms can be classified as convergecast-based algorithms, flooding algorithms, local algorithms, and cluster-based algorithms.

Flooding Algorithms Flooding algorithms flood whatever data is available at each node. This is very expensive since it communicates all the data. It is trivial to develop algorithms for certain functions which flood all the data and guarantee correctness of the results. In most interesting cases, we strive to develop algorithms which can do better than this naive approach.

Convergecast Algorithms The second category, convergecast algorithms, are also extremely simple. In such algorithms the computation takes place over a spanning tree. For any peer, computing the network sum, for example, amounts to receiving the data of all the neighbors, then adding its value and propagating the result to its parent. Algorithms such as [156] provide generic solutions — suitable for the computation of multiple functions. They are also extremely communication efficient: computing the average, for instance, only requires one message from each peer. The problem with these algorithms is that they are extremely synchronized — with every round of computation taking a lot of time. This becomes very problematic when the data is dynamic and computation has to be iterated frequently due to data change.

Local algorithms Local algorithms are a class of highly efficient algorithms developed for P2P networks. They are data dependent distributed algorithms. However, in a distributed setup data dependency means that at certain conditions peer can cease to communicate with one another and the algorithm terminates with the same result compared to a centralized scenario. These conditions can occur after a peer has collected the statistics of just few other peers. In such cases, the overhead of every peer becomes independent of the size of the network — either a small constant or a slow growing polynomial compared to the network size. Furthermore, these data dependent conditions can be rechecked every time the data or the system, changes. If the change is stationary (i.e., the result of the computation remains the same then, very often, no communication is needed. This fea-

ture makes local algorithms exceptionally suitable for P2P networks as well as for wireless sensor networks.

The idea of using local rules for algorithms dates back to the seventies. John Holland described such rules for non-linear adaptive systems and genetic algorithms in his seminal work for biological systems [80]. Local evolutionary rules for grid-based cellular automaton were first introduced in 1950's by John Von Neumann [139] and later adopted in many fields such as artificial agents, VLSI testing, physical simulations to mention a few. In the context of graph theory, local algorithms were used in the early nineties. Linial [112] and later Afek *et al.* [1] talked about local algorithms in the context of the graph coloring problems. Naor and Stockmeyer [133] asked what properties of a graph can be computed in constant time independent of the graph size. Kutten and Peleg [108] have introduced local algorithms for fault-detection in which the cost depends only on the unknown number of faults and not on the entire graph size. They have developed solutions for some key problems such as the maximal independent set (MIS) and graph coloring. Kuhn *et al.* [106] have suggested that some properties of graphs cannot be computed locally.

Local algorithms for P2P data mining include the majority voting and association rule mining protocol developed by Wolff and Schuster [196]. This algorithm and the ones discussed in this section guarantee *eventual correctness* — when the computation terminates, each node computes the correct result compared to a centralized setting. In the most simple form, majority rule protocol deals with the following computation: suppose each node has two real numbers x_i and y_i, and the goal is to find out if $\sum_{i=1}^{n} x_i \geq \sum_{i=1}^{n} y_i$, where there are n peers in the network. This protocol is eventually correct, fault-tolerant and robust to data and network changes. They are efficient as well since in the typical dynamic setup they require far less resources compared to broadcast. Based on its variants, researchers have further proposed more complicated algorithms such as facility location [105], outlier detection [19], and meta-classification [116].

More recently, Bhaduri and Kargupta [13] have proposed a local algorithm for deterministic multivariate regression in P2P networks. The idea is to use a two-step approach. A local algorithm is used to track the fitness of the current regression model and the data. If the data has changed such that the model no longer fits the data, a feedback loop is used and the model is rebuilt. Experimental results show the accuracy and low monitoring cost of the proposed technique.

All the algorithms depicted above require the existence of a communication tree to avoid duplicate accounting of data. Some work have shown that they can be transformed to work for general networks as well [14]. Note that, as shown in [14], such a tree can be efficiently constructed and maintained using variations of Bellman-Ford algorithms [57][84]. Generic local algorithms have also been proposed for a wide variety of data mining problems such as L2-norm thresholding, k-means monitoring and more [195].

Although several local algorithms have been developed, theoretical description of local algorithm complexity is still an open issue. Birk *et al.* [14] have proposed a new metric *Veracity Radius (VR)* to capture the locality of local algorithms. This radius counts the number of hops which is necessary in order to get the correct result from a local algorithm. However this framework works only for simple aggregate problems such as majority voting.

While most of the work in local algorithms focus on developing algorithms which compute models based on all the peers' data, Ping *et al.* [116] have proposed a different method. In that work, a meta classification algorithm is described in which every peer computes a weak classifier on its own data. Then, weak classifiers are merged into a meta classifier by computing — per new sample — the majority of the outcomes of the weak classifiers. The computation of weak classifiers requires no communication overhead at all, and the majority is computed using the majority voting protocol [196].

Another work which should be mentioned in the context of local algorithms is the

WildFire algorithm by Bawa *et al.* [11]. WildFire too makes use of local techniques to prune messages. It further suggests an algorithm for the problem of Means Monitoring (computation of Sum or Average). However, the algorithm presented there is very costly for dynamic data.

Cluster-based Algorithms There is an intrinsic relation between local algorithms and communication-efficient algorithms for large clusters. While the latter algorithms rely on broadcasting [164][168], or on hierarchical communication patterns [8] they still have the idea that not all of the processors need to send their entire data for the global result to be computed accurately. For instance, the association rule mining algorithms for clusters [164] and P2P networks [196] both rely on an efficient majority votes. The algorithm presented by Sharfman *et al.* [168], generalizes [164]; but still relies on broadcast as the communication model. Additionally, the notions of data and network changes throughout the algorithm execution are missing from the work on clusters, possibly because the smaller scale of those systems permit ignoring or recovering from rare failures.

2.7 Related work: Distributed Data Stream Mining

There exists a plethora of work in the area of distributed data stream mining. Not only have the distributed data mining and databases community contributed to the literature, a bulk of the work also comes from the wireless and sensor networks community. In this section we discuss some of the related papers with pointers for further reading.

Computation of complex functions over the union of multiple streams have been studied widely in the stream mining literature. Gibbons *et al.* [67] presents the idea of doing coordinated sampling in order to compute simple functions such as the total number of ones in the union of two binary streams. They have developed a new sampling strategy to sample from the two streams and have shown that their sampling strategy can reduce the

space requirement for such a computation from $\Omega(n)$ to $log(n)$, where n is the size of the stream. Their technique can easily be extended to the scenario where there are more than two streams. The authors also point out that this method would work even if the stream is non-binary (with no change in space complexity).

Much work has been done in the area of query processing on distributed data streams. Chen *et al.* [32] have developed a system 'NiagaraCQ' which allows answering continuous queries in large scale systems such as the Internet. In such systems many of the queries are similar. So a lot of computation, communication and I/O resources can be saved by properly grouping the similar queries. NiagaraCQ achieves the same goal. Their grouping scheme is incremental and they use an adaptive regrouping scheme in order to find the optimal match between a new query and the group to which the query should be placed. If none of these matches, then a new query group is formed with this query. The paper does not talk about reassignment of the existing queries into the newly formed groups, rather leaves it as a future work.

An different approach has been described by Olston *et al.* [142]. The distributed model described there has nodes sending streaming data to a central node which is responsible for answering the queries. The network links near the central node becomes a bottleneck as soon as the arrival rate of data becomes too high. In order to avoid that, the authors propose installing filters which restrict the data transfer rate from the individual nodes. Node O installs a filter of width or range W_O and of range $[L_O, H_O]$. W_O is centered around the most recent value of the object V ($L_O = V - \frac{W_O}{2}$ and $H_O = V + \frac{W_O}{2}$). Now the node does not send updates if V is inside the range $L_O \leq V \leq H_O$; otherwise it sends updates to the central node and recenters the bounds L_O and H_O. This technique provides the answers to queries approximately and works in the circumstances where we do not require the exact answer to the queries. Since in many cases the user can provide the query precision that is necessary, the filters can be made to work after setting the bounds

based on this user input.

The sensor network community provides a rich literature on the data stream mining algorithms. Since, in many applications, the sensors are deployed in hostile terrains, one of the most fundamental task aims at developing a general framework for monitoring the network themselves. A similar idea has been presented in [205]. This paper presents a general framework and shows how decomposable functions like min, max, average, count and sum can be computed over such an architecture. The architecture is highlighted by three tools that the authors call *digests, scans* and *dumps*. *Digests* are the network parameters (*e.g.* count of the number of nodes) that are computed either continuously, periodically or in the event of a trigger. *Scans* are invoked when the *digests* report a problem (*e.g.* a sudden drop in the number of nodes) to find out the energy level throughout the network. These two steps can guide a network administrator towards the location of the fault which can be debugged using the *dumps* (dump all the data of a single or few of the sensors). Furthermore, this paper talks about is the distributed computing of some aggregate functions (mean, max, count etc.). Since all these functions are decomposable, the advantage is in-network aggregation of partial results up a tree. The leaf does not need to send all its data to the root and in this way vital savings can be done in terms of communication. The major concern though is maintaining this tree structure in such a dynamic environment. Also this technique would fail for numerous non-decomposable functions *e.g.* median, quantile etc.

The above algorithm describes a way of monitoring the status of the sensor network itself. There are many data mining problems that need to be addressed in the sensor network scenario. Such an algorithm for multi-target classification in the sensor networks has been developed by Kotecha *et al.* [103] Each node makes local decisions and these decisions are forwarded to a single node which acts as the manager node. The maximum number of targets is known in apriori, although the exact number of targets is not known in advance. Nodes that are sufficiently apart are expected to provide independent feature

vectors for the same target which can strengthen the global decision making. Moreover, for an optimal classifier, the number of decisions increases exponentially with the number of targets. Hence the authors propose the use of sub-optimal linear classifiers. Through real life experiments they show that their suboptimal classifiers perform as well as the optimal classifier under mild assumptions. This makes such a scheme attractive for low power, low bandwidth environments.

Frequent items mining in distributed streams is an active area of research. There are many variants of the problem that has been proposed in the literature. Interested readers are referred to [118] for a description. Generally speaking there are m streams $S_1, S_2, .., S_m$. Each stream consists of items with time stamps $< d_{i1}, t_{i1} >, < d_{i2}, t_{i2} >$, etc. Let S be the sequence preserving union of all the streams. If an item $i \in S$ has a count $count(i)$ (the count may be evaluated by an exponential decay weighting scheme). The task is to output an estimate $\widehat{count}(i)$ of $count(i)$ whose frequency exceeds a certain threshold. Each node maintains a precision threshold and outputs only those items exceeding the precision threshold. As two extreme cases, the threshold can be set to very low (≈ 0) or very high (≈ 1). In the first case, all the intermediate nodes will send everything without pruning resulting in a message explosion at the root. In the second case, the intermediate nodes will send a low number of items and hence no more pruning would be possible at the intermediate nodes. So the precision selection problem is crucial for such an algorithm to produce meaningful results with low communication overhead. The paper presents a number of ways to select the precision values (they call it precision gradients) for different scenarios of load minimization.

2.8 Related work: Data Mining in Sensor Networks

In this section we present some related work in the area of data mining in sensor networks.

Wireless Sensor Networks (WSN's) are finding increasing number of applications in many domains, including battle fields, smart buildings, and even the human body. Most sensor networks consist of a collection of light-weight (possibly mobile) sensors connected via wireless links to each other or to a more powerful gateway node that is in turn connected with an external network through either wired or wireless connections. Currently most of the sensor nodes are connected in a hierarchical fashion, with partial or complete centralized control. Simplicity and low maintenance of P2P architecture suggests that the next generation of sensor nodes will communicate in an P2P architecture using ad-hoc links.

Hence, data mining in wireless sensor networks (WSN's) is related to data processing over a P2P network. However there are some significant differences. In many cases, while designing algorithms for a P2P environment, we are mainly concerned with reducing the communication and making the algorithm scalable. However, in a typical WSN we not only need to consider communication overhead, but we need to understand the energy requirement of the algorithm, the number of idle cycles that each sensor may waste due to computation/communication, the size of the message payload and possibly more. So a highly communication efficient algorithm in a P2P setting that requires each node to constantly probe for incoming messages is not a very suitable candidate in a WSN environment. More details about information processing in sensor network can be found in the book by Zhao and Guibas [204].

Many researchers have focused on making the data collection task in a sensor network energy efficient. LEACH, LEACH-C, LEACH-F [76][77], and PEGASIS [111] are some of the attempts towards making that. Intrusion detection in sensor networks has received considerable attention in the last few years mainly because of the nature of the task that typical sensor networks are intended for. Radivojac *et al.* [157] has developed an algorithm for intrusion detection in a supervised framework, where there are far more negative instances than positive (intrusions). An unsupervised approach to the outlier detection problem in

sensor networks is presented by Palpanas et. al. [146], where kernel density estimators are used to estimate the distribution of the data generated by the sensors, and then the outliers are detected depending on a distance based criterion. More recently, Branch et. al. [19] proposed another unsupervised approach to detect outliers in wireless sensor networks using local algorithms. This algorithm is robust to data and network changes and, unlike many other local algorithms, works for any general network topology.

In many cases computing decomposable aggregates such as sum, average, count is facilitated by doing an in network aggregation since these these aggregates can be computed locally and combined to produce the final correct solution. However there are many function such as median, quantiles and the like whose exact computation requires knowledge of the entire data. Greenwald *et al.* [70] presents a general framework for computing the ϵ-approximate quantile and median of the sensor data. The make the aggregates "quasi"-decomposable and thereby achieve excellent reduction in communication per node. The experimental results show the effectiveness of their approach.

An important area of research is clustering the sensors themselves since nodes that are clustered together can easily communicate with each other or solve a spatial problem. In many cases it is posed as an optimization problem. Ghiasi *et al.*, [64] presents the theoretical aspects of this problem with special emphasis to energy optimization. Each cluster is represented by a master node (similar to the centroid in a *k*-means clustering technique). Their clustering algorithm ensures that each cluster is balanced and the total distance between the sensor nodes and the master nodes is minimized. Several other techniques are also presented in the literature such as [200][34].

A recent workshop on "Data Mining in Sensor Networks" contains several relevant works: an architecture for data mining [17], artificial neural-network algorithms [107], pre-processing using EM [42] and more.

2.9 Related work: Information Retrieval in P2P Networks

Information retrieval in P2P networks (P2PIR)is closely related to data mining in P2P networks. Searching and ranking in P2P file sharing networks form basic foundation of P2PIR. P2PIR is considered a special sub branch of distributed information retrieval. Callan [25] presents an overview of distributed information retrieval where the searchable databases are distributed across the network and there is no central coordinator or index to aid in the searching process. An detailed study of the different issues and prospects of P2PIR is presented by Sia [171]. Search and ordering in P2P networks are active areas of research. In the remainder of this section we discuss the work related to search and ranking in P2PIR.

2.9.1 Search in P2P Network

Several techniques have been proposed in the literature for efficient searching in P2P networks. Lv *et al.* [117] argues that the flooding-based search technique currently employed by Gnutella [68] is not scalable due to its high bandwidth consumption in a large P2P network. The alternate ideas proposed in the paper are to use (1) a TTL-token based approach and (2) a random walk based approach. Each node in the network forwards a query message, called walker, randomly to one of its peers. To reduce the query response time, the idea of the 1-walker is extended to a k-walker, where k independent walkers are simultaneously propagated. The paper shows through simulations that their technique achieves comparable results (compared to Gnutella) while reducing the network traffic by two orders of magnitude. A different approach was proposed by Kalogeraki et. al [91]. The proposed technique, known as Random Breadth First Search (RBFS), works by forwarding a search message to only a fraction of its peers, selected at random. Since a node is able to make decisions based on only a subset of the nodes in the network, the advantage of RBFS is that it does not require global knowledge and is therefore scalable. However, being prob-

abilistic in nature, the results are true based on the results of the theory of statistics. Yang *et al.* [198] present a collection of heuristics for efficient searching in P2P networks. Iterated deepening, directed BFS and local indices are the three techniques proposed. Experimental results show that all of these techniques have their own merits and demerits and they are applicable for the next generation P2P networks.

Given a collection of text documents, the problem of retrieving the subset that is relevant to a particular query has been studied extensively [161][159]. Till today, one of the most successful techniques for addressing this problem is the vector space ranking model originally proposed by Salton and Yang [161]. Cuenca-Acuna and Nguyen [36] proposed *PlanetP*, a P2P searching and information retrieval system based on the vector space ranking model [161]. The main problem addressed in [36] is how to search for and retrieve documents relevant to a query posed by some member of a *PlanetP* community. The basic idea in *PlanetP* is that each community member creates an inverted (word-to-document) index of the documents (IDF) that it wishes to share, summarizes this index in a compact form, and diffuses the summary throughout the community. More specifically, the paper shows how to approximate TFxIDF using compact summaries of individual peers' inverted indexes rather than the inverted index of the entire communal store. This is important for the scalability of the algorithm. The goal is to construct a content addressable publish/subscribe service that uses gossiping of local state across unstructured communities for information retrieval. To achieve this *PlanetP* uses a heuristic for adaptively determining the set of peers that should be contacted for a query.

A popular technique of disseminating information among the nodes of a large P2P network is Distributed Hash Tables (DHT). Distributed hash tables (DHTs) are a class of decentralized distributed systems that provide a lookup service similar to a hash table: (name, value) pairs are stored in the DHT, and any participating node can efficiently retrieve the value associated with a given name. The hash table is not kept at a central site, rather

this lookup information is disseminated among the nodes, in such a way that a change in the set of participants causes a minimal amount of disruption. This allows DHTs to scale to extremely large numbers of nodes and to handle continual node arrivals, departures, and failures. Because DHTs support efficient querying and replication, they can be used for distributed file systems, P2P file systems and content distribution systems.

Several DHTs have been proposed in the literature. CHORD [175], CAN [158], PASTRY [160] and TAPESTRY [203] are some of the most popular ones. DHT's are characterized by two quantities — keyspace partitioning and overlay network. Most DHTs use some variant of consistent hashing to map keys to nodes. A function $\delta(k_1, k_2)$ is used to define the distance between two keys k_1 and k_2 which is unrelated to geographical distance or network latency. Each node is assigned a single key called its *identifier* (ID). A node with ID i owns all the keys for which i is the closest ID, measured according to δ. For example, Chord DHT views the keys as if they are arranged on a circle and $\delta(k_1, k_2)$ is the distance traveling clockwise around the circle from k_1 to k_2. The main advantage with consistent hashing is that removal or addition of one node changes only the set of keys owned by the nodes with adjacent IDs, and leaves all other nodes unaffected. Along with this, each node maintains a set of links to some other nodes in the overlay network. The overlay network in a DHT topology is such that for any key k, a node either owns k or knows a node which is closer to k. Given this, routing a message/query is easy: forward the message to the neighbor whose ID is closest to k. When no such neighbor exists, we have reached the owner of the data. DHT's are scalable and the cost of inserting and deleting a node grows very slowly.

Several other P2P search techniques and systems have been developed. Bessonov *et al.* [12] proposed an open architecture for P2P searching as part of the OASIS project. ODISSEA (Open DIStributed Search Engine Architecture) was proposed in [180]. Tang *et al.* [186] proposed another P2P search algorithm called pSearch, that distributes document

indices through the P2P network based on document semantics generated by Latent Semantic Indexing (LSI). Klampanos and Jose [101] proposed another architecture for search in P2P networks where there exists semi-collaborating peers in which peers collaborate to achieve some global goal. However their own proprietary data is protected. Yee and Frieder [199] have looked at several existing P2P search techniques and proposed a combination which is generally very effective in practice.

2.9.2 Ranking in P2P Network

P2P search forms the basis of information extraction from P2P networks. In many cases, the search results are huge and unordered. In this section we discuss several algorithms that have been proposed in the literature for ranking the results in order of relevance before presenting it to the user. However, ordering the results is a challenging task in absence of a centralized document index. The centralized page rank algorithm, used by Google[4], and its variants are not applicable for a distributed scenario like P2P network. As a result, top-k items identification in distributed settings and its variants are actively researched even today.

Fagin *et al.* [54] proposed a threshold algorithm for ordering top-k queries. Cao *et al.* [28] later showed that the algorithm presented in [54] consumed a lot of bandwidth when the number of nodes is large. The paper presents a new algorithm called "Three-Phase Uniform Threshold" (TPUT). TPUT reduces network bandwidth consumption by pruning away ineligible objects, and terminates in three round-trips regardless of data input.

The KLEE framework [122] proposed by Michel *et al.* presents a novel family of algorithms for top-k query processing in wide-area distributed environments. It shows how efficiency can be enjoyed at low result-quality penalties. Shi et. al [170] presented a distributed version of the page-ranking algorithm used by Google to rank results obtained in a

[4]http://en.wikipedia.org/wiki/PageRank

P2P network. Their Pagerank found by their algorithm converges to the centralized Pagerank algorithm. However, their technique is only applicable to structure P2P overlay networks. Nejdl *et al.* [136] considered the problem of ranking and finding the top-k queries from a schema-based P2P network such as Edutella [137]. Edutella is a P2P infrastructure for storing and retrieving RDF (Resource Description Format) metadata in a distributed environment. The proposed distributed algorithm makes use of local rankings, rank merging and optimized routing based on peer ranks, and minimizes both answer set size and network traffic among peers. They use the super peer approach to route the queries which may be absent in many emerging P2P infrastructures. In 2005 [6], the authors improved upon their earlier work and proposed a nearest match, or top-k query in P2P networks based on the same super peer backbone. The proposed technique uses progressive improvement of query results, thereby reducing network traffic by a lot. Ranking the best peers based on a set of multiple queries has been addressed by Tempich *et al.* [187].

2.10 Related work: Data Mining in GRID

Grid can be defined as - "the ability, using a set of open standards and protocols, to gain access to applications and data, processing power, storage capacity and a vast array of other computing resources over the Internet" [71]. Grid computing has gained popularity as a distributed computational resource for a plethora of massive computational tasks which is difficult to handle by a single computational resource. Grids find applications in diverse domains such as:

1. earth science applications (funded by NSF and NASA) — climate/weather modeling, earthquake simulation and more,
2. financial applications such as financial modeling,
3. genetic algorithms using the Condor service,

4. biological domains to study the effect of protein folding,

5. E-sciencE project, which is based in the European Union and includes sites in Asia and the United States, is a follow up project to the European DataGrid (EDG) and is arguably the largest computing grid on the planet,

6. and many more.

Grid computing was popularized by Ian Foster, Carl Kesselman and Steve Tuecke in their seminal work [59] who are widely recognized as the “father of the modern grids” [193]. A Grid is a type of parallel and distributed system that enables the sharing, selection, and aggregation of resources distributed across multiple administrative domains based on the resources availability, capacity, performance, cost and users’ ”quality-of-service requirements”. Since many disciplines of science involve scavenging through massive volumes of data, computational grids offer a perfect platform for various data mining tasks. Grid systems differ from P2P systems in the sense that grid systems have a built-in centralized control structure and network infrastructure which assigns jobs and ensures optimal system performance. P2P systems on the other hand do not rely on such tightly coupled central authority. Nevertheless, since both Grid and P2P systems deal with heterogenous distributed resources coupled together to solve a common goal, they both relate to the idea of distributed data mining and share some commonalities in terms of the data mining constraints. Talia and Trunfio [183] discuss the similarities between Grid and P2P computing. In another work, Talia and Skillicorn [182] argues that the grid offers unique prospects for mining of large data sets due to its collaborative storage, bandwidth and computational resources. Cannataro *et al.* [26] address general issues in distributed data mining over the grid. Since the grid uses resource from various sources, proper resource allocation to different jobs is very important and challenging job. Hoschek *et al.* [81] discusses the data management issues for grid data mining. Several interesting ongoing grid projects involve

data mining over the grid. The NASA Information Power Grid [5], Papyrus [4], the Data Grid [38], the Knowledge Grid [27] are to name a few of them. The Globus Consortium has put forward the Globus Toolkit, which is open source and helps researchers with grid computing.

2.11 Related work: Distributed AI and Multi-Agent Systems

In this section we give a very brief overview of the field of distributed artificial intelligence (DAI) and multi-agent systems (MAS). For a detailed overview, interested readers are referred to the PhD thesis of Stone [177][178] and the review paper by Sen [166].

The evolution of MAS can be traced to the early work in AI such as the blackboard system [140], Minsky's *Society of Mind* concept [125] and more. They fostered the concept of independent entities working in a parallel manner to achieve some common goal.

In early years of MAS, Victor Lesser with the help of his colleagues and students developed the distributed vehicle monitoring testbed [50][109]. Almost at the same time Agha [2] and Hewitt [79] proposed the actors framework for parallel communication of the agents. The contract net protocol developed by Smith [173] specifies agent communication protocols.

Several researchers have looked into the following problems in MAS/DAI research. Distributed search in scheduling and constraint satisfaction was dealt with by Durfee [52]. Multiagent learning is an active area of research. Haynes and Sen [75] have written as series of papers which talk about learning in multi-agent learning.

There are several ways of dividing the MAS and the field of DAI [45][51]. Here we follow the taxonomy presented by Stone [177]. The MAS literature can be subdivided into the following four subsections as described below.

[5] http://www.gloriad.org/gloriad/projects/project000053.html

2.11.1 Homogenous and non-communicating MAS

In homogenous MAS, all the agents are same in functionality, domain knowledge and possible actions. They differ mainly in terms of their inputs (and hence outputs) because they are located at different physical locations in the world. The agents cannot communicate directly.

Using homogenous non-communicating agents, Balch and Arkin [5] have investigated the formation control for robot teams. Such agents have also been used elsewhere: local knowledge [167], stigmergy [69] and recursive modeling [49].

2.11.2 Heterogenous and non-communicating MAS

In this scenario the agents are different in their goals, actions and domain knowledge. The assumption that the agents do not communicate directly still holds. The goals of the agents can be such that they are cooperative or disruptive. This issue of benevolence vs. competitiveness has been addressed by Goldman and Rosenschein [69]. Game theoretic formulations for cooperative games such as prisoner's dilemma have been proposed by Mor and Rosenschein [127]. Littman [114] considered a zero sum non-cooperating game and proposed Minimax-Q which is a variation of Q-learning.

The use of heterogenous non-communicating agents extends to competitive co-evolution [75], adaptive load balancing [162] and more.

2.11.3 Homogenous and communicating MAS

In homogenous communicating, the agents are the same (except the inputs and hence output) as in homogenous non-communicating, except that these agents can now talk to each other. The communication can be broadcast, point-to-point or shared memory. This area is closely related to the sensor network domain. Use of MAS in this domain include the cooperative distributed vision project [135] and the real time traffic information system

[128].

2.11.4 Heterogenous and communicating MAS

This is perhaps the most difficult scenario to model because of heterogeneity and communication. Several techniques have been proposed for heterogenous agent communication such as KQML [56]. A plethora of work exists with this type of MAS such as cooperative co-evolution [24], multi-agent Q-learning [184], query response in information networks [181], coalitions [169] and more. Interested readers are refereed to Stone [177] for a detailed discussion on this topic.

2.12 Applications of Peer-to-Peer Data Mining

The proliferation of the P2P networks over the last decade has generated a large number of interesting applications. In this section we barely scratch the surface. For a more detailed discussion interested readers are refereed to the P2P article in Wikipedia [144].

2.12.1 File storage

The biggest and probably the most widely used application of P2P networks is file storage. The past decade has witnessed a large number of P2P storage systems both academic and commercial. On the academic side CHORD [175], PASTRY [160] and CAN [158] are some of the well known P2P file storage applications. Commercial P2P storage systems such as Napster [134], Gnutella [68], BitTorrents [16] and Kazaa [97] have been extremely popular for data storage, music, video and photo sharing. More upcoming P2P storage applications include the UbiStorage project [190]. Hasan *et al.* [74] presents a detailed overview of the different P2P distributed file storage systems.

These file storage applications offer an important testbed for P2P data mining algorithms. Data mining algorithms are necessary for recommender systems, search engines,

classification and clustering of the documents/files in such settings.

2.12.2 Education

Several interesting P2P projects are currently being undertaken at different educational institutions or in a consortium of such institutions.

One of them is the ScienceNet P2P search engine, currently under deployment by the Karlsruhe Institute of Technology - Liebel-Lab [165]. The idea is to do distributed indexing of university or research websites to keep up-to-date information about the collaborators and researchers. It is based on "YaCy", a P2P search engine software. Universities, research institutes or individuals can download the free java software and install it on their own machine. The latter would then join the SceienceNet network and help in the indexing of researchers or universities.

With the rapid explosion of data, many universities are collaborating on data storage using a P2P technology. One such endeavor is by the Pennsylvania State University, MIT and Simon Fraser University who are carrying on a project called LionShare [113] designed for facilitating file sharing among educational institutions globally.

Another project worth mentioning is the PlanetLab project originally started by the researches at UC Berkeley, Intel labs and Princeton [151]. The project aims at maintaining a large number of nodes as a testbed for running applications for distributed systems. As of October 2007, there 825 nodes spread across the US, Europe and Asia. Most of the participants are educational institutions, research labs and corporate agencies interested in distributed and networking systems research. Many research papers published today contain simulation results based on PlanetLab — a sure sign that this testbed is becoming popular.

2.12.3 Bioinformatics

Bioinformatics is the interaction of Biology and Information Science or Data Mining. Being a data rich subject, bioinformatics is a natural candidate for application of P2P mode of computing. P2P networks are specifically used to run large programs which can identify drug candidates such as the one conducted at the Centre for Computational Drug Discovery at Oxford University in cooperation with the National Foundation for Cancer Research, stared from 2001. Chinook [35][126] is a P2P bioinformatic service focusing on exchange of analysis between bioinformatic researchers worldwide. It consists of self-administered programs for computational biologists to run and compare various bioinformatics software. Tranche [188] is an bioinformatics network developed to ease the data sharing problems among the researchers.

2.12.4 Consumer Applications

P2P technologies have found a number of uses in our everyday lives. Real-time audio and video streaming using the Peer Distributed Transfer Protocol (PDTP) is more common today than simple offline audio and video downloading. One such implementation is the "distribustream" peercasting service [47]. Many TV channels are now broadcast using the P2P technology, commonly referred to as P2PTV. Sopcast [174], developed in China is an extremely popular P2PTV service which people use worldwide for viewing multi-language TV channels in TV resolution. Joost [88] is also another hugely popular service.

P2P technology is actively used in tele-communications. Today, many Voice-over-IP (VoIP) service is handled in a P2P fashion — Skype [172] being one of the most popular. Currently it has over fifty million users.

Sun Microsystems has developed the JXTA™ [90] protocol specifically to allow developers develop P2P applications. Numerous P2P chat applets and instant messaging service exist today based on JXTA™.

2.12.5 Sensor Network Applications

A completely different application of P2P networks is the sensor network application originally started by the US Defense Agency (DARPA) for battlefield surveillance. Many sensor network-based civilian applications have emerged such as habitat monitoring, healthcare applications, home automation, and traffic control.

Sensors are deployed in hostile locations to provide detailed environmental information. These sensors form an ad-hoc P2P connection. For sensors, the preservation of battery power is critical due to two reasons — (1) the sensors are small in size having limited power and (2) due to their physical placement, it is not possible to change the batteries often. In such sensors, wireless communication significantly consumes more energy than any other operation. Therefore, algorithms designed for such networks need to be extremely communication-efficient and put the sensors into a sleep mode to conserve the battery power. Moreover, sensors need to operate in a router-less environment with no global IPs. Finally, due to the limited power and hostile environment, light-weight, wireless sensor networks are highly dynamic with sensor dropout and edge failure a common occurrence.

2.12.6 Mobile Ad-hoc Networks (MANETs)

MANETs are getting increasing attention in many wireless application domains. Several data rich environments (e.g. vehicular ad-hoc networks, P2P e-commerce environments) are emerging and they are likely to need data analytic supports for efficient and personalized services. Consider a simple profiling application launched via cellular phones that tries to automatically connect to MANET like network formed by different cellular phones in the vicinity and identify peers with similar interest to form a social network. A lightweight P2P classification algorithm may be very handy in such an application.

Many upcoming vehicular networks are built using wireless communication technol-

ogy enabling cars to talk to each other to get traffic information, suggest alternate routes, alert drivers in case of approaching bad weather and prevent life-threatening accidents by taking evasive actions. These cars form an ad hoc P2P connection; node failures and re-connection are very high. All these techniques use variants of distributed data mining algorithms to generate models on the fly and in real-time. For example, predicting a future action requires one to model the past behavior and current state using classification or regression algorithms. The Campus Vehicular Testbed at the University of California Los Angeles (CVet@UCLA) [191] is one such initiative.

2.13 Summary

In this chapter first we have presented a detailed background study of the different types of P2P networks. We have classified P2P networks into three main categories — (1) centralized, (2) decentralized but structured, and (3) decentralized and unstructured. This type of classification has been proposed by Lv *et al.* [117]. Since P2P data mining algorithms operate in many different environments, we have presented a few important and desired properties of P2P data mining algorithms. A comprehensive literature review of the previous work related to this area of research is presented next. For convenience, we have divided the related work into the following areas:

1. distributed data mining
2. data mining in P2P networks
3. data stream monitoring
4. data mining in sensor networks
5. information retrieval in P2P networks
6. data mining in Grids

7. machine learning in multi-agent systems and distributed artificial intelligence

Finally we have presented some application areas of P2P data mining which, over the last decade, have received considerable attention.

Chapter 3

DEFRALC: A DETERMINISTIC FRAMEWORK FOR LOCAL COMPUTATION IN LARGE DISTRIBUTED SYSTEMS

3.1 Introduction

In sensor networks, P2P systems, grid systems, and other large distributed systems there is often the need to model the data that is distributed over the entire system. In most cases, centralizing all or some of the data is a costly approach. The cost becomes higher if the data distribution changes and the computation needs to be updated to follow that change.

Statistical models can be built and updated from distributed data in several different ways. The *global* computation paradigm uses the information from all the nodes in the network to construct a data mining model by aggregating all the data to a central location. Such a computing model has several advantages: (1) a large variety of models can be computed, (2) the output can be provably correct for many problems, (3) the algorithms can be very simple (e.g. broadcast, convergecast). However their major drawback is scalability. Computing models using the information from all the nodes in the network means the algorithms scale proportional to the size of the network. This becomes severely problematic especially for large networks consisting of millions of nodes. Moreover, these algorithms

are very synchronous, unlikely to be useful in today's asynchronous P2P networks. On the other hand, it should be noted that global computation is not always essential. Consider for example the problem of finding the mean of an array of numbers where each node has one number. Instead of transmitting the entire data, each node can simply communicate with its neighbors and then the result can be propagated. Such algorithms which compute the results in a small neighborhood and then propagate the results are called *local* algorithms. Local algorithms, introduced in Chapter 2, are highly efficient family of algorithms developed for distributed systems. Firstly, local algorithms are in-network algorithms in which the computation nodes only exchange statistics — not raw data — with their neighbors to compute the result. Secondly, at the heart of a local algorithm there is a data dependent criteria dictating when nodes can avoid sending updates to their neighbors. An algorithm is generally called local if the total communication per node is upper bounded by some constant, independent of the number of nodes in the network. Primarily for this reason, local algorithms exhibit high scalability. Moreover local algorithms exhibit higher fault tolerance due to their ability to localize and isolate the faults and avoid a single point of failure.

The superior scalability of local algorithms motivates us to develop a local framework for data mining in large distributed environments. Specifically, we focus on local algorithms which are provably correct compared to a centralized execution. Our goal in this chapter is to first introduce the reader with some definitions and properties of local algorithms. Next we present our *DeFraLC* framework (**De**terministic **F**ramework for **L**ocal **C**omputing). Section 3.6 presents the main theorem in *DeFraLC*. Section 3.7 discusses the **Pe**er-to-Peer **Ge**neric **M**onitoring **A**lgorithm (*PeGMA*) in *DeFraLC* capable of computing complex functions of distributed data. Finally we also discuss several instantiations of the *PeGMA* and also demonstrate how the *DeFraLC* can be used for developing several data mining algorithms.

3.2 What is locality ?

In this section we present a brief intuition of local algorithms as used in this dissertation. Consider a large network in which every node has some data and our task is to compute some function defined on the union of all the data of all the peers. One way of doing this computation is to allow each peer to communicate with all the other nodes in the network to get a global model. This process, reminiscent of gossip, flooding or broadcast-style communication is too expensive — typically the communication cost per node is O(size of the network). As a result, increasing the network size increases the communication cost by the same factor. Therefore these algorithms are not suitable for modern-day large-scale P2P networks spanning possibly millions of peers.

An alternate approach is to develop distributed algorithms which satisfy the following property: each node should base its output by communicating only with a fixed number of neighbors and the total size of the query exchanged for the entire execution of the algorithm should also be bounded. Ideally, these "fixed bounds" should be small constants, independent of the size of the system. As a consequence, these algorithms exhibit locality in their execution and are therefore highly scalable. Increasing the system size has virtually no effect on the performance of these algorithms making them ideal candidate for P2P networks. Moreover, locality has intrinsic relation to fault tolerance and robustness — for any peer since the execution is local, only peers in its fixed neighborhood are affected.

An example of local communication is shown in Figure 3.1. In the figure, peer 47 is connected to peers 25, 30, 31, 45 and 49. If the algorithm run on this network is such that each peer can do this local computation and still guarantee some global result, it would eventually mean high scalability of such algorithms.

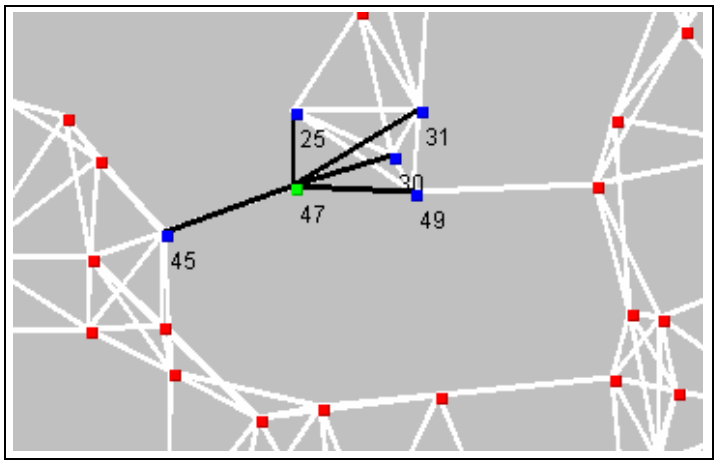


FIG. 3.1. Figure showing connections to immediate neighbors.

3.3 Local Algorithms: Definitions and Properties

The idea of using local rules for algorithms dates back to the seventies. John Holland described such rules for non-linear adaptive systems and genetic algorithms in his seminal work for biological systems [80]. Local evolutionary rules for grid-based cellular automaton were first introduced in 1950's by John Von Neumann [139] and later adopted in many fields such as artificial agents, VLSI testing, physical simulations to mention a few. In the context of graph theory, local algorithms were used in the early nineties. Linial [112] and later Afek *et al.* [1] talked about local algorithms in the context of the graph coloring problems. Kutten and Peleg [108] have introduced local algorithms for fault-detection in which the cost depends only on the unknown number of faults and not on the entire graph size. They proposed solutions for some key problems such as the maximal independent set (MIS) and graph coloring. Naor and Stockmeyer [133] defined locality as follows.

Definition 3.3.1 (Local Algorithm: Naor and Stockmeyer [133])**.** *An algorithm is called* ***local*** *if it runs in constant time, independent of the size of the network.*

This definition implies that if an algorithm runs in constant time t, then it must com-

pute its results by contacting nodes only in a neighborhood of radius t. It further argues that local algorithms offer good fault tolerance — a failure at a node n can only affect nodes in the neighborhood of n. However in many cases, strict independence is not always possible. For example, if the computation of a node increases very slowly compared to the size of the network, then the algorithm is not local according to this definition. However we feel that it should be called local since its growth rate is bounded, even though not independent of the size of the network.

Graph theoretic formulations of locality and network computations are discussed extensively in the book by Peleg [149]. The locality-sensitive approach to network computations tries to restrict computations within a particular region (cluster). Intuitively, the idea is to cover the network by overlapping clusters and each node participates in computation in its own cluster. As a result the cost of a task depends only on the locality properties of the cluster.

The application of local algorithms for data mining in P2P networks is relatively new. Data mining in P2P networks requires algorithms that work in dynamic environments. The following is a definition of local algorithm proposed by Wolff *et al.* [195].

Definition 3.3.2 (Local Algorithm: Wolff et al [195])**.** *An algorithm is **local** if there exists a constant c such that for any number of peers n, there are instances (i.e., neighbors Γ_i, input and intermediate states for each peer P_i) such that from the last change of the data and system to the termination state of the algorithm no peer uses more than c additional messages, memory bits, CPU cycles, and time units.*

Although the above definition is strictly from an efficiency standpoint, it is fairly easy to argue the effectiveness of such a definition from the standpoint of the graph theoretic notions of *locality*. According to Definition 3.3.2, for any graph with unit delay along its edges, a peer can contact at most c peers in its entire neighborhood for it to be local. This means that the result of a query can propagate at most c hops. In other words the data

propagation and the running time is bounded by a constant c for any graph G.

However, Definition 3.3.2 is not without its limitations. First of all in order for a distributed algorithm to be termed local, the communication per node should be small. Requiring it to be a constant is not sufficient, since the constant can itself be large with respect to the size of the network. Moreover, the size of the message is not bounded. To deal with this, Datta *et al.* [39] proposed a definition of local algorithm which is later refined by Das *et al.* [37]. First we define the α-neighborhood of a vertex.

Definition 3.3.3 (α-neighborhood of a vertex)**.** *Let $G = (V, E)$ be the graph representing the network where V denotes the set of nodes and E represents the edges between the nodes. The α-**neighborhood of a vertex** $v \in V$ is the collection of vertices at distance α or less from it in G: $\Gamma_v(\alpha, v, V) = \{u | dist(u, v) \leq \alpha\}$, where $dist(u, v)$ denotes the length of the shortest path in between u and v and the length of a path is defined as the number of edges in it.*

Next we define an α-local query based on the definition of α-neighborhood of a vertex.

Definition 3.3.4 (α-local query)**.** *Let $G = (V, E)$ be a graph as defined in last definition. Let each node $v \in V$ store a data set X_v. An α-**local query** by some vertex v is a query whose response can be computed using some function $\mathcal{F}(X_\alpha(v))$ where $X_\alpha(v) = \{X_v | v \in \Gamma_v(\alpha, v, V)\}$.*

Finally we present the definition of (α, γ)-local algorithm.

Definition 3.3.5 ((α, γ)-local algorithm)**.** *An algorithm is called **(α, γ)-local** if it never requires computation of a β-local query such that $\beta > \alpha$ and the total size (measured in bits) of the response to all such α-local queries sent out by a peer is bounded by γ. α can be a constant or a function parameterized by the size of the network while γ can be parameterized by both the size of the data of a peer and the size of the network.*

We call such an (α, γ)-local algorithm ***efficient*** if both α and γ are either small constants or some slow growing functions (sub-linear) with respect to its parameters.

Consider an algorithm with broadcast as the communication mode. For such an algorithm, γ is O(*size of the network*). Hence broadcast-based algorithms are not local according to this definition.

Next consider an algorithm which does convergecast over a communication tree in order to compute a sum of the numbers held at each peer. The protocol works as follows. A leaf peer sends its data to its parent up the tree. Whenever an intermediate peer gets data from all its childern except one, it adds all the data it has received from its children and its own data. Then it sends it to the neighbor from whom it has not received anything. If it has received from everyone, it computes the sum of all those including its own data and then broadcasts it. Otherwise, a peer keeps waiting. The following lemma (Lemma 3.3.1) shows that such an algorithm is not local considering bounded γ.

Lemma 3.3.1. *The convergecast algorithm for computing the sum (as stated above) is not* $(O(1), \gamma)$*-local for a constant* γ *or slow growing* γ *with respect to the size of the network* n.

Proof. It is obvious that α=1 since, for any peer, communication takes place among its immediate neighbors only.

For bounds on γ, let the data at each peer be d_i and let m bytes be needed to represent d_i. For any peer P_i, let $Child_i$ denote the size of the subtree i.e. number of children rooted at P_i. P_i does the following computation when its gets data from all its neighbors:

$$d_i + \sum_{\ell \in \Gamma_i - 1} d_\ell$$

The number of bytes necessary to represent this sum is

$$m + m \times (Child_i) = m \times (1 + Child_i)$$

For the root node, $Child_i = n - 1$, therefore the message size at the root is $m \times n$. Thus there exist one peer in the network, the root whose $\gamma = O(n)$. Hence the convergecast algorithm for computing the sum is not $(0(1), \gamma) - local$ for a constant or slowly growing γ. □

However, using such a convergecast technique for computing the maximum of the numbers located at each peer is $(O(1), \gamma)$-local when γ is a constant, independent of the size of the network. Lemma 3.3.2 proves this claim.

Lemma 3.3.2. *The convergecast algorithm for computing the maximum is $(O(1), \gamma)$-local, when γ is a constant, independent of the size of the network.*

Proof. It is obvious that α=1 since, for any peer, communication takes place among its immediate neighbors only.

For bounds on γ, let the data at each peer be d_i and let m bytes be needed to represent d_i. Peer P_i does the following computation:

$$\max\{d_i, \max_{\ell \in \Gamma_i - 1}\{d_\ell\}\}$$

At any node, the maximum can be represent using m bytes. Thus for any node in the network, the total size of the query is a constant m. Hence the convergecast algorithm for computing the maximum is $(O(1), \gamma) - local$, for a constant γ. □

Although the previous definition discusses about the efficiency of local algorithms, it does not specify the quality of the result. The yardstick for measuring the result of any distributed data mining algorithm for computing a function $\mathcal{F}$ is to compare it with a centralized algorithm for computing the same function $\mathcal{F}$ having access to all of the data and the resources. Traditionally, based on accuracy, local algorithms can broadly be classified as: *exact* and *approximate*. In an exact local algorithm, once the computation terminates, the result computed by each peer is the same as that compared to a centralized execution

e.g. [196][195][105]. On the other hand, researchers have also proposed approximate local algorithms using probabilistic techniques e.g. K-means [40] and top-l elements identification [37].

Before we formally define the accuracy of local distributed data mining algorithms, we state what we mean by the accuracy of a centralized algorithm under the same conditions.

Definition 3.3.6 (Accuracy of centralized algorithm)**.** *Given a distributed algorithm for computing a function $\mathcal{F} : \mathbb{R}^d \rightarrow \mathbb{O}$, (where $\mathbb{O}$ has an arbitrary range) the **accuracy of a centralized algorithm** for computing $\mathcal{F}$ is the accuracy achieved by an algorithm having access to all of the data and other distributed resources at a central location.*

We are now in a position to discuss the accuracy of (α, γ)-local distributed data mining algorithms.

Definition 3.3.7 ((α, γ)-probably approximately correct local algorithm)**.** *Let $\boldsymbol{E}\,(A, \mathcal{F})$ denote the error induced by an algorithm A when computing a function $\mathcal{F}$ as measured against the true $\mathcal{F}$. Let X be a random variable which measures the difference of the error between an (α, γ)-local distributed algorithm A_d computing $\mathcal{F}$ and a centralized algorithm A_c, computing the same function $\mathcal{F}$. A_d is (ϵ, δ) **correct**, if*

$$Pr(X > \epsilon) < 1 - \delta$$

In this work, the local algorithms that we have developed as part of the *DeFraLC* framework are *exact*. We focus on developing *local* algorithms which satisfy the following desired properties:

- It should guarantee eventual correctness.
- It should have a local stopping rule — each peer should be able to determine independently when to stop communicating its data.

- Any peer should ideally communicate with only its immediate neighbors — at the most it can communicate with peers which are at a small distance (bounded number of hops) from it.

- The total communication overhead per peer should be a small number and independent of the size of the network.

Low communication overhead makes local algorithms highly scalable. Computation for *local* algorithms is, in most cases, independent of the size of the system. So we can potentially increase the size of the system with very little increase in communication and deterioration in quality. Henceforth we will use the (α, γ)-locality definition as the general definition of local algorithms.

3.4 Approach

Let $\mathcal{G}$ denote a collection of data tuples (input vectors) that is horizontally distributed over a large (undirected) network of machines (peers) wherein each peer communicates only with its immediate neighbors (one hop neighbors) in the network. The communication network can be thought of as a graph with vertices (peers) V.

The algorithms we describe in this dissertation (*PeGMA* and its instantiations) deal with deterministic efficient local function computation defined on the global dataset $\mathcal{G}$. Note that vectors in $\mathcal{G}$ can be combined in many different ways to produce a statistic: weighted linear combination and its variants (sum, difference, average), cartesian product, inner product, cross product and more. Linear combination of the vectors in $\mathcal{G}$ provide a good measure of the variability of the vectors in $\mathcal{G}$. Thus, the algorithms described here focus on function computation defined on linear combination of vectors in $\mathcal{G}$. Furthermore, the algorithms are correct when compared to a centralized scenario and this correctness can be guaranteed even in the case of data and network changes. Efficiency of our technique is

due to the use of local data dependent rules which allows a peer to get the correct result by communicating in a small neighborhood and thereby cutting down the number of messages drastically.

Linear combinations such as the average find use in many data mining and signal processing problems such as source detection and localization [150], recommender systems in the Internet, leader election and many more. As we further show in this chapter and the next chapter, function computation on the global average vector is a very powerful tool — a wide variety of complex data mining algorithms such as k-means, eigen-monitoring, multivariate regression and decision trees can be developed from using a such a simple yet powerful primitive.

The next section gives a formal description of the notation and problem definition.

3.5 Notations, Assumptions, and Problem Definition

This section we discuss the notations and assumptions which will be used throughout the rest of this chapter.

3.5.1 Notations

Let $V = \{P_1, \ldots, P_n\}$ be a set of peers (we use the term peers to describe the peers of a peer-to-peer system, motes of a wireless sensor network, etc.) connected to one another via an underlying communication infrastructure such that the set of P_i's neighbors, Γ_i, is known to P_i. Additionally, P_i is given a time varying set of input vectors in $\mathbb{R}^d$, $S_i = \left\{\overrightarrow{x_i^1}, \overrightarrow{x_i^2}, \ldots\right\}$. In this distributed data mining literature this is often referred to as the horizontally data distribution scenario.

In this work peers communicate with one another by sending sets of input vectors as defined in Definition 3.5.1.

Definition 3.5.1 (Message for sets)**.** *The **message** that peer P_i needs to send to P_j consists of a set of vectors and is denoted by $X_{i,j}$. Each vector is in $\mathbb{R}^d$ and the size of the set depends on the data that peer P_i needs to send to P_j.*

Below, we show that for our purposes statistics such as the average and number of vectors are sufficient. Assuming reliable messaging, once a message is delivered both P_i and P_j know both $X_{i,j}$ and $X_{j,i}$. Our goal is to compute functions defined on the union of all the data of all the peers. We want to achieve this without all peers requiring to send all of their data. For this, we define four sets of vectors which are local to a peer. As shown in a later section (Section 3.6), conditions on these local vectors imply certain conditions on the global dataset and hence functions defined on the union of all data of all peers.

Definition 3.5.2 (Knowledge)**.** *The **knowledge** of P_i, denoted as $\mathcal{K}_i$, is the union of S_i with $X_{j,i}$ for all $P_j \in \Gamma_i$ and is defined as $\mathcal{K}_i = S_i \cup \bigcup_{P_j \in \Gamma_i} X_{j,i}$.*

Definition 3.5.3 (Agreement)**.** *The **agreement** of P_i and any of its neighbors P_j is $\mathcal{A}_{i,j} = X_{i,j} \cup X_{j,i}$.*

Definition 3.5.4 (Withhled knowledge)**.** *The **withheld knowledge** of P_i with respect to a neighbor P_j is the subtraction of the agreement from the knowledge $\mathcal{W}_{i,j} = \mathcal{K}_i \setminus \mathcal{A}_{i,j}$.*

Definition 3.5.5 (Global input)**.** *The **global input** at any time is the set of all inputs of all the peers and is denoted by $\mathcal{G} = \bigcup_{i=1,\ldots,n} S_i$.*

Many data mining primitives in the literature can be posed as a geometric problem. For example consider the problem of checking if the L2 norm of a vector in $\mathbb{R}^2$ is greater than a user-defined ϵ. This has applications in many complex data mining problems as we show in this dissertation. Geometrically, checking if the L2 norm is greater than ϵ amounts to checking if the vector lies inside a circle of radius ϵ. The theorems in the *DeFraLC* framework deal with convex regions and their collections in order to guarantee results on

the global data (and hence functions defined on them) based on the sets of local vectors only *viz.* $\mathcal{K}_i$, $\mathcal{A}_{i,j}$ and $\mathcal{W}_{i,j}$. Hence, next we define convex region, invariant functions defined in convex regions and ω-cover i.e. C_ω.

Definition 3.5.6 (Convex region)**.** *A region $R \subseteq \mathbb{R}^d$ is **convex**, if for every two points $\overrightarrow{x}, \overrightarrow{y} \in R$ and every $\alpha \in [0,1]$, the weighted average $\alpha \overrightarrow{x} + (1-\alpha)\overrightarrow{y} \in R$.*

Definition 3.5.7 (Invariant function in a convex region)**.** *Let $\mathcal{F}$ be a function from $\mathbb{R}^d \to \mathbb{0}$. We say $\mathcal{F}$ is **invariant** in R (in short $\mathcal{F}_R$-invariant), if the value of $\mathcal{F}$ on R is constant, i.e. $\forall \overrightarrow{x}, \overrightarrow{y} \in R : \mathcal{F}(\overrightarrow{x}) = \mathcal{F}(\overrightarrow{y})$.*

Definition 3.5.8 (ω-cover of a domain for a function $\mathcal{F}$)**.** *A collection of non-overlapping convex regions $C_\omega = \{R_1, R_2, \ldots, R_n\}$ is a ω-cover of $\mathcal{F}$ if:*

1. *every $R_i \in C_\omega$ is convex and $\mathcal{F}_{R_i}$-invariant in R_i*
2. *the area of $\mathbb{R}^d$ not covered by $\bigcup_{i=1}^{n} R_i$ is less than ω*

Our next definition deals with the tie regions.

Definition 3.5.9 (Tie region)**.** *Let $C_\omega = \{R_1, R_2, \ldots, R_n\}$ be a convex cover for a function $\mathcal{F}$ as defined earlier. Also let $D = \left\{\overrightarrow{x} \middle| \overrightarrow{x} \in \mathbb{R}^d\right\}$ be the set of all vectors in $\mathbb{R}^d$. The **tie region** T is the region uncovered by $C = \bigcup_{i=1}^{n} R_i$ i.e. $T = \{\overrightarrow{x} | \overrightarrow{x} \in D \setminus C\}$.*

Next we define the global termination state of a distributed asynchronous algorithm.

Definition 3.5.10 (Global termination state of a distributed asynchronous algorithm [10])**.** *A distributed asynchronous algorithm is said to have terminated globally or reached a **termination at a certain global state** if*

1. *all nodes become idle,*
2. *all edges become empty (i.e. all messages are delivered), and*
3. *the system does not change for a time period Δ_t, which is parameterized by the diameter of the network and the maximal edge delay of a node.*

3.5.2 Assumptions

In this section we state the assumptions that we have made for developing algorithms under *DeFraLC*.

Assumption 3.5.1. *In this dissertation we are interested in computing functions defined on linear combinations of vectors in $\mathcal{G}$. Hence the underlying space is a linear vector space of $\mathcal{G}$.*

Linear combinations such as the average is a technique for combining vectors in $\mathcal{G}$. Average vector also measures the variability of the vectors in the set $\mathcal{G}$. Simplicity combined with the ability to represent a number of well-known data mining problems (as we show later) motivates us to concentrate in such a linear space.

Assumption 3.5.2. *Communication among neighboring peers is **reliable** and **ordered**.*

This assumption can easily be enforced using standard numbering and retransmission (in which messages are numbered, ordered and retransmitted if an acknowledgement does not arrive in time), ordering, and heart-beat mechanisms. Moreover, this assumption is not uncommon and have been made elsewhere in the distributed algorithms literature [63]. Khilar and Mahapatra [100] discuss the use of heartbeat mechanisms for failure diagnosis in mobile ad-hoc networks.

Assumption 3.5.3. *Communication takes place over a **communication tree**.*

It is assumed that sets of vectors sent from P_i to P_j is never sent back to P_j. Since we are dealing with statistics of data (e.g. counts), one way of ensuring this is to make sure that a tree overlay structure is imposed on the original network such that there are no cycles. As described in Section 3.5.3, this assumption allows us to send statistics of the sets and not the sets themselves. We could get around this assumption in one of two ways. (1) Liss *et al.* [14], have shown how the distributed majority voting algorithm (which

require a communication tree) can be developed for general networks. We could use a similar technique here, although the generalization to other networks may be non-trivial (2) The underlying tree communication topology could be maintained independently of our framework using standard techniques such as [63] (for wired networks) or [110] (for wireless networks).

Our goal is to develop a local framework for correctly computing a function $\mathcal{F}$ defined on $\mathcal{G}$ (the same function output at each node). However, the network is dynamic in the sense that the network topology can change (peers may enter or leave at any time) or the data held by each peer can change. Hence $\mathcal{G}$, the union of all peers data, can be thought of as time-varying as well as the set of neighbors Γ_i for each peer P_i. This leads us to our next assumption.

Assumption 3.5.4. *Peers are notified on **changes** (addition/deletion of data tuples or modification to the attribute values) in their own data S_i, and in the set of their neighbors Γ_i.*

Assumption 3.5.5. *Input data tuples (or vectors) in S_i are **unique**.*

Assumption 3.5.6. *A ω-**cover** C_ω can be **precomputed** for every $\mathcal{F}$ and desired ω.*

Note that Assumption 3.5.6 is the weakest of all the assumptions. In this work (see Section 3.7.3) we show how such a ω-cover can be devised for a specific function — the hyper-sphere in R^d — which solves several data mining problems. Finding a cover C_ω for an arbitrary $\mathcal{F}$ in $\mathbb{R}^d$ depends on $\mathcal{F}$ and the space $\mathbb{R}^d$. The communication complexity of our algorithm might suffer due to an inefficient choice of C_ω. Moreover, for the hyper-sphere, we show in our extensive experiments that the communication complexity increases linearly with the increase in d, while the space increases exponentially. Formulating an algorithm which can output the set ω-cover for all functions $\mathcal{F}$ defined in $\mathbb{R}^d$ is, according to us, an open research topic.

3.5.3 Sufficient Statistics

The algorithms we describe in this dissertation deal with computing functions of linear combinations of vectors in $\mathcal{G}$ i.e. we restrict ourselves to a linear vector space. For clarity, we will focus on one such combination — the average. When approximating the average — due to the assumption of a tree overlay used in this work — we can completely represent the sets $\mathcal{G}$, S_i, $\mathcal{K}_i$, $\mathcal{A}_{i,j}$, $X_{i,j}$ and $X_{j,i}$ we previously defined with some of their statistics: the average vector of the set and its size. We can do that because as long as messages sent from P_i to P_j are independent of $X_{j,i}$, both $X_{i,j} \cap X_{j,i} = \emptyset$ and $\mathcal{W}_{i,j} \cap \mathcal{K}_j = \emptyset$. Next we define the average and the size of any arbitrary set.

Definition 3.5.11 (Average vector of a set)**.** *Given a set X, the **average vector of the set** is the average of all the vectors in the set and is denoted by $\overrightarrow{X}$.*

Definition 3.5.12 (Size of a set)**.** *Given a set X, the **size of the set** is the cardinality of the set i.e. the number of vectors in the set and is denoted by $|X|$.*

Definition 3.5.13 (Average vector of a set union)**.** *Given two sets X and Y, the **average vector of the set** $X \cup Y$, is defined as follows: $\overrightarrow{X \cup Y} = \frac{\overrightarrow{X}+\overrightarrow{Y}}{2}$.*

Definition 3.5.14 (Size of a set union)**.** *Given two sets X and Y, the **size of the set** $X \cup Y$, is defined as follows: $|X \cup Y| = |X| + |Y|$.*

Similar statistics can be written for each of the following sets: $S_i, \mathcal{K}_i, \mathcal{A}_{i,j}$ and $\mathcal{W}_{i,j}$. Given the definitions of the average vectors and the sizes of the sets, the idea is to express the statistics of $\mathcal{K}_i$, $\mathcal{A}_{i,j}$, and $\mathcal{W}_{i,j}$ in terms of the statistics of S_i and $X_{i,j}$'s rather than in terms of the statistics of unions thereof. Below we define the average vectors and sizes of each of the sets.

Definition 3.5.15 (Average and size of local dataset)**.** *The **size** and **average** of **local dataset** S_i are defined as:*

- *Size:* $|S_i|$ *= number of data tuples in* S_i
- *Average:* $\overrightarrow{S_i} = \frac{1}{|S_i|} \sum_{\overrightarrow{x} \in S_i} \overrightarrow{x}$

Definition 3.5.16 (Average and size of knowledge)**.** *The* ***size*** *and* ***average*** *of* ***knowledge*** $\mathcal{K}_i$ *are defined as:*

- *Size:* $|\mathcal{K}_i| = |S_i| + \sum_{P_j \in \Gamma_i} |X_{j,i}|$
- *Average:* $\overrightarrow{\mathcal{K}_i} = \frac{|S_i|}{|\mathcal{K}_i|} \overrightarrow{S_i} + \sum_{P_j \in \Gamma_i} \frac{|X_{j,i}|}{|\mathcal{K}_i|} \overrightarrow{X_{j,i}}$

Definition 3.5.17 (Average and size of agreement)**.** *The* ***size*** *and* ***average*** *of* ***agreement*** $\mathcal{A}_{i,j}$ *are defined as:*

- *Size:* $|\mathcal{A}_{i,j}| = |X_{i,j}| + |X_{j,i}|$
- *Average:* $\overrightarrow{\mathcal{A}_{i,j}} = \frac{|X_{i,j}|}{|X_{i,j}|+|X_{j,i}|} \overrightarrow{X_{i,j}} + \frac{|X_{j,i}|}{|X_{i,j}|+|X_{j,i}|} \overrightarrow{X_{j,i}}$

Definition 3.5.18 (Average and size of withheld knowledge)**.** *The* ***size*** *and* ***average*** *of* ***withheld knowledge*** $\mathcal{W}_{i,j}$ *are defined as:*

- *Size:* $|\mathcal{W}_{i,j}| = |\mathcal{K}_i| - |\mathcal{A}_{i,j}|$
- *Average:* $\overrightarrow{\mathcal{W}_{i,j}} = \frac{|\mathcal{K}_i|}{|\mathcal{W}_{i,j}|} \overrightarrow{\mathcal{K}_i} - \frac{|\mathcal{A}_{i,j}|}{|\mathcal{W}_{i,j}|} \overrightarrow{\mathcal{A}_{i,j}}$ *or* $\overrightarrow{\mathcal{W}_{i,j}} = \overrightarrow{0}$ *in case* $|\mathcal{W}_{i,j}| = 0$

Definition 3.5.19 (Average and size of global knowledge)**.** *The* ***size*** *and* ***average*** *of* ***global knowledge*** $\mathcal{G}$ *are defined as:*

- *Size:* $|\mathcal{G}| = \sum_{i=1}^{n} |S_i|$
- *Average:* $\overrightarrow{\mathcal{G}} = \frac{1}{|\mathcal{G}|} \sum_{i=1}^{n} \overrightarrow{S_i}$

3.5.4 Problem Definition

Problem 1: Given a function $\mathcal{F} : \mathbb{R}^d \rightarrow \mathbb{O}$ and a time varying data set S_i at each peer P_i, we are interested in computing the value of $\mathcal{F}\left(\overrightarrow{\mathcal{G}}\right)$ at each peer P_i.

Considering the dynamic and distributed nature of the data, accurate computation of $\mathcal{F}\left(\overrightarrow{\mathcal{G}}\right)$ can be very difficult. This might be, for example, because $\overrightarrow{\mathcal{G}}$ changes at a rate faster than the rate at which data can be propagated through the network. We therefore pose a quality standard and a correctness standard suitable for this scenario.

Definition 3.5.20 (Accuracy)**.** *Given an algorithm for computing $\mathcal{F}\left(\overrightarrow{\mathcal{G}}\right)$ the **accuracy** of the algorithm is measured as the fraction of peers which compute the correct outcome $\mathcal{F}\left(\overrightarrow{\mathcal{G}}\right)$ at any time t.*

Definition 3.5.21 (Eventually correct)**.** *An algorithm which converges to a termination state is **eventually correct** if it guarantees that at any termination state every peer computes the correct outcome $\mathcal{F}\left(\overrightarrow{\mathcal{G}}\right)$, where the correct output is the one which is computed on all the data of all the peers.*

3.5.5 Illustration

Having stated the notations and problem definition, in this section we present an illustrative example.

Let there be two peers P_1 and P_2 connected to each other such that $\Gamma_1 = P_2$ and $\Gamma_2 = P_1$. The local data of P_1 and P_2 i.e. S_1 and S_2 are vectors in $\mathbb{R}^2$. Let

- $S_1 = \{(2, 6), (3, 8), (1, 4)\}$
- $S_2 = \{(7, 9), (10, 12)\}$

Furthermore for simplicity we consider an initialization state of the peers whereby $X_{1,2} = X_{2,1} = \emptyset$.

We first calculate the knowledge, agreement and witheld knowledge of P_1:

- Sets for P_1:
  - $\mathcal{K}_1 = S_1 \cup \bigcup_{P_2 \in \Gamma_1} X_{2,1} = \{(2,6),(3,8),(1,4)\}$
  - $\mathcal{A}_{1,2} = X_{1,2} \cup X_{2,1} = \emptyset$
  - $\mathcal{W}_{1,2} = \mathcal{K}_1 \setminus \mathcal{A}_{1,2} = \{(2,6),(3,8),(1,4)\}$
- Statistics for P_1:
  - $|S_1| = 3$
  - $\overrightarrow{S_1} = \frac{1}{|S_1|} \sum_{\overrightarrow{x} \in S_1} \overrightarrow{x} = \{(2,6)\}$
  - $|\mathcal{K}_1| = |S_1| + \sum_{P_2 \in \Gamma_1} |X_{2,1}| = 3$
  - $\overrightarrow{\mathcal{K}_1} = \frac{|S_1|}{|\mathcal{K}_1|}\overrightarrow{S_1} + \sum_{P_2 \in \Gamma_1} \frac{|X_{2,1}|}{|\mathcal{K}_1|}\overrightarrow{X_{2,1}} = \{(2,6)\}$
  - $|\mathcal{A}_{1,2}| = |X_{1,2}| + |X_{2,1}| = 0$
  - $\overrightarrow{\mathcal{A}_{1,2}} = \frac{|X_{1,2}|}{|X_{1,2}|+|X_{2,1}|}\overrightarrow{X_{1,2}} + \frac{|X_{2,1}|}{|X_{1,2}|+|X_{2,1}|}\overrightarrow{X_{2,1}} = \emptyset$
  - $|\mathcal{W}_{1,2}| = |\mathcal{K}_1| - |\mathcal{A}_{1,2}| = 3$
  - $\overrightarrow{\mathcal{W}_{1,2}} = \frac{|\mathcal{K}_1|}{|\mathcal{W}_{1,2}|}\overrightarrow{\mathcal{K}_1} - \frac{|\mathcal{A}_{1,2}|}{|\mathcal{W}_{1,2}|}\overrightarrow{\mathcal{A}_{1,2}} = \{(2,6)\}$

Similarly for P_2, we can write:

- Sets for P_2:
  - $\mathcal{K}_2 = \{(7,9),(10,12)\}$
  - $\mathcal{A}_{2,1} = \emptyset$
  - $\mathcal{W}_{2,1} = \{(7,9),(10,12)\}$
- Statistics for P_2:
  - $|S_2| = 2$

- $\overrightarrow{S_2} = \{(8.5, 10.5)\}$
- $|\mathcal{K}_2| = 2$
- $\overrightarrow{\mathcal{K}_2} = \{(8.5, 10.5)\}$
- $|\mathcal{A}_{2,1}| = 0$
- $\overrightarrow{\mathcal{A}_{2,1}} = \emptyset$
- $|\mathcal{W}_{2,1}| = 2$
- $\overrightarrow{\mathcal{W}_{2,1}} = \{(8.5, 10.5)\}$

Now the global knowledge $\mathcal{G} = \{(2, 6), (3, 8), (1, 4), (7, 9), (10, 12)\}$ and the statistics are:

- $|\mathcal{G}| = 5$
- $\overrightarrow{\mathcal{G}} = \{(4.6, 7.8)\}$

We may be interested in finding if $\overrightarrow{\mathcal{G}}$ lies inside a square centered at (0,0) and side of length 16 as shown in Figure 3.2. The output is 0 if $\overrightarrow{\mathcal{G}} \in R_s$ and 1 otherwise. Also shown in the figure are $C_\omega = \{R_s, R_1, R_2, R_3, R_4\}$, $\overrightarrow{\mathcal{K}_1}$, $\overrightarrow{\mathcal{K}_2}$ and $\overrightarrow{\mathcal{G}}$. It is evident from the figure that $\overrightarrow{\mathcal{G}} \in R_s$ and $\overrightarrow{\mathcal{K}_1} \in R_s$ but $\overrightarrow{\mathcal{K}_2} \notin R_s$. Therefore the correct answer is 0. Peer P_1 will output 0 and P_2 will output 1. The algorithms we describe in this dissertation enable every peer to produce the correct outcome as $\overrightarrow{\mathcal{G}}$ by exchanging only the statistics and not $\overrightarrow{S_1}$ and $\overrightarrow{S_2}$.

For a problem in $\mathbb{R}^d$, the knowledge, agreement and witheld knowledge can be evaluated in a similar fashion. However, it becomes significantly more difficult to find the cover C_ω for an arbitrary $\mathcal{F}$ and $\mathbb{R}^d$. The communication complexity of our algorithm might suffer due to an inefficient choice of C_ω.

The next section presents a theorem (Theorem 3.6.1) which allows a peer to decide if $\overrightarrow{\mathcal{G}}$ resides inside a convex region in C_ω based on only local information i.e. its knowledge, agreement and witheld knowledge.

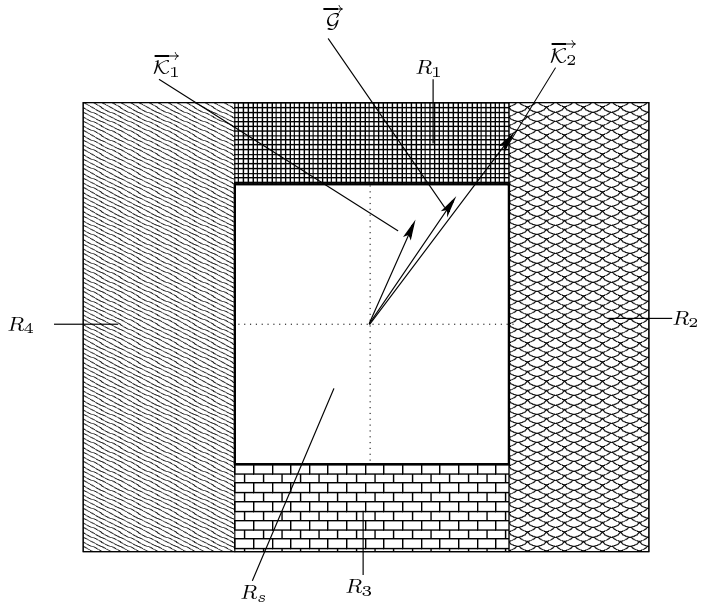


FIG. 3.2. An example showing the cover and the vectors $\vec{\mathcal{G}}$, $\vec{\mathcal{K}_1}$ and $\vec{\mathcal{K}_2}$. Each shaded region is a convex region along with the unshaded region inside the square.

3.6 Main Theorems

The main theorems of the *DeFraLC* framework apply to the eventual correctness of local algorithms. Since eventual correctness is concerned with termination states, the state of the algorithm can be expressed in terms of only the variables of each peer. Local algorithms rely on conditions which permit extending the rules defined on the states of the different peers onto rules defined on the global data $\mathcal{G}$. Specifically, the theorems described herein state conditions on $\overrightarrow{\mathcal{K}_i}$, $\overrightarrow{\mathcal{A}_{i,j}}$, and $\overrightarrow{\mathcal{W}_{i,j}}$, subject to which it can be determined if $\overrightarrow{\mathcal{G}}$ resides inside a specific convex region $R_i \in C_\omega$.

Definition 3.6.1 (Unification of peers)**.** *Let P_i and P_j be two neighboring peers. The process of deleting P_j, by combining its knowledge $\overrightarrow{\mathcal{K}_j}$, agreement $\overrightarrow{\mathcal{A}_{j,i}}$ and witheld knowledge $\overrightarrow{\mathcal{W}_{j,i}}$ with P_i is known as* ***unification****.*

Theorem 3.6.1. [Convex Stopping Rule] *Let $P_1, \ldots, P_n$ be a set of peers connected to each other via a communication tree $G(V, E)$. Let S_i, $X_{i,j}$, $\mathcal{G}$, $\mathcal{K}_i$, $\mathcal{A}_{i,j}$, and $\mathcal{W}_{i,j}$ be as defined in the previous section. Let R be an arbitrary convex region in $\mathbb{R}^d$. If at time t no messages traverse the network, and for each P_i, $\overrightarrow{\mathcal{K}_i} \in R$ and for every $P_j \in \Gamma_i$ $\overrightarrow{\mathcal{A}_{i,j}} \in R$, and for every $P_j \in \Gamma_i$ either $\overrightarrow{\mathcal{W}_{i,j}} \in R$ or $\mathcal{W}_{i,j} = \emptyset$, then $\overrightarrow{\mathcal{G}} \in R$.*

Proof. Assume an arbitrary P_i sends all of the vectors in $\mathcal{W}_{i,j}$ to its neighbor P_j and receives $\mathcal{W}_{j,i}$. The new knowledge of P_i is $\mathcal{K'}_i = \mathcal{K}_i \cup \mathcal{W}_{j,i}$. The average vector of the new knowledge is $\overrightarrow{\mathcal{K'}_i} = \overrightarrow{\mathcal{K}_i \cup \mathcal{W}_{j,i}}$. If $\mathcal{W}_{j,i} = \emptyset$ then of course $\mathcal{K'}_i = \mathcal{K}_i$. Otherwise, since in a tree $\mathcal{W}_{j,i} \cap \mathcal{K}_i = \emptyset$, $\mathcal{K'}_i$ can be rewritten as a weighted average of the two vectors $\overrightarrow{\mathcal{K'}_i} = \alpha\overrightarrow{\mathcal{K}_i} + (1-\alpha)\overrightarrow{\mathcal{W}_{j,i}}$ for some $\alpha \in [0, 1]$. Since both $\overrightarrow{\mathcal{K}_i}$ and $\overrightarrow{\mathcal{W}_{j,i}}$ are in R, and since R is convex, any average vector is also in R. It follows that $\overrightarrow{\mathcal{K'}_i}$ is in R too.

The new withheld knowledge of P_i with respect to any other neighbor P_k, $\mathcal{W'}_{i,k}$, is equal to the withheld knowledge prior to the unification, plus the added vectors, $\mathcal{W'}_{i,k} = \mathcal{W}_{i,k} \cup \mathcal{W}_{j,i}$. Again, $\mathcal{W}_{j,i} = \emptyset$ then $\mathcal{W'}_{i,k} = \mathcal{W}_{i,k}$. Otherwise, since both $\overrightarrow{\mathcal{W}_{i,k}}$ and $\overrightarrow{\mathcal{W}_{j,i}}$ are in R and since $\mathcal{W}_{j,i} \cap \mathcal{W}_{i,k} = \emptyset$, it follows that $\overrightarrow{\mathcal{W}_{i,k} + \mathcal{W}_{j,i}} = \alpha\overrightarrow{\mathcal{W}_{i,k}} + (1-\alpha)\overrightarrow{\mathcal{W}_{j,i}}$ is in R too.

Note that $\mathcal{A}_{i,k}$ does not change because no message is sent from P_i to P_k or vice-versa. It follows that P_j can be eliminated by connecting all of its other neighbors to P_i, and by setting the agreement of P_i and its new neighbor to the previous agreement that neighbor had with P_j.

Similarly consider P_z — any other neighbor of P_i. Using a similar argument, P_z can be unified with P_i and hence deleted. If this process is continued, we will eventually be left with a single peer P_i with $\overrightarrow{\mathcal{K}_i} = \overrightarrow{\mathcal{G}}$. Since in every step of the induction $\overrightarrow{\mathcal{K}_i}$ remained inside R, $\overrightarrow{\mathcal{G}}$ must be in R. □

Theorem 3.6.1 can be explained using Figure 3.3. The top row of the figure shows the state of the peers P_1, P_2, P_3 and P_4 at any time instance. Note that all the vectors are inside

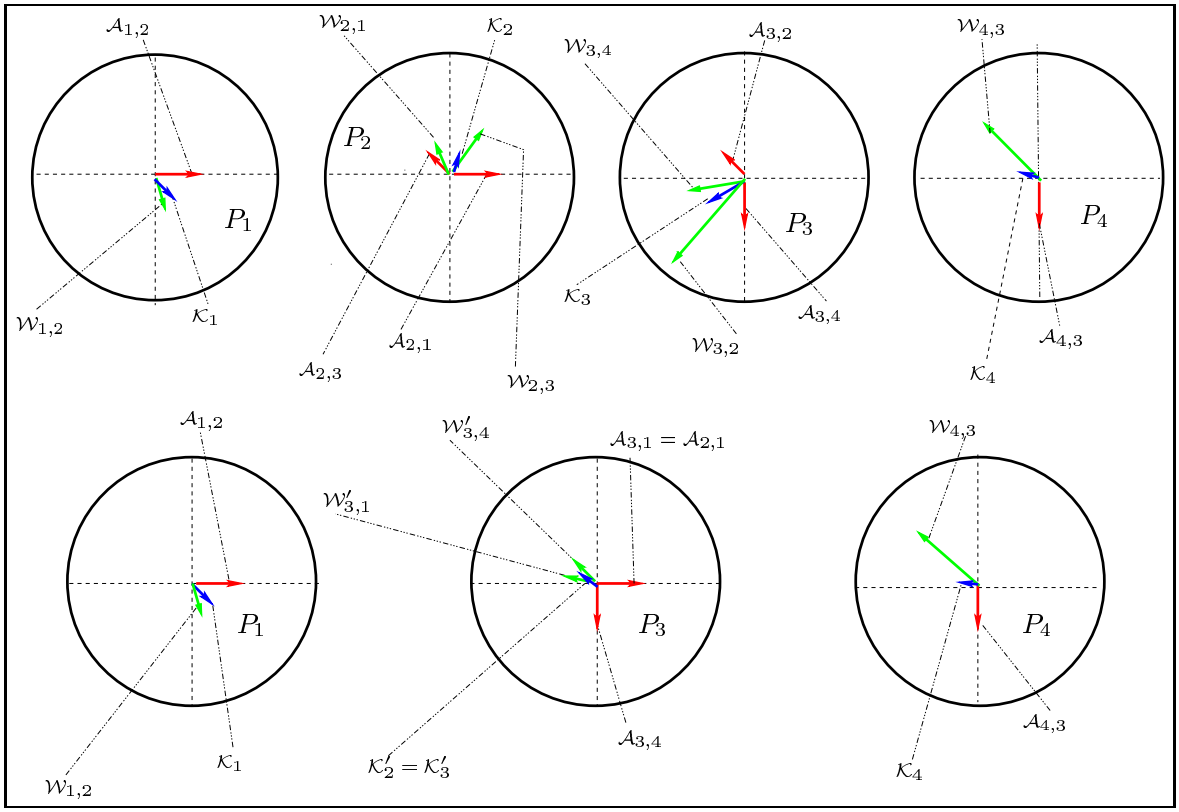


FIG. 3.3. Vectors of four peers before and after Theorem 3.6.1 is applied. The knowledge, the agreement and the withheld knowledge are shown in the figure. The top row represents the initial states of the four peers. The bottom row depicts what happens if P_2 sends all its data to P_3.

some R (in this case, a circle). Let us suppose that P_2 and P_3 exchange all of their data. The new knowledge of P_3, $\overrightarrow{\mathcal{K}'_3}$, is a convex combination of $\overrightarrow{\mathcal{K}_3}$ and $\overrightarrow{\mathcal{W}_{2,3}}$. The new withheld knowledge of P_3 with respect to P_4 is denoted by $\overrightarrow{\mathcal{W}'_{3,4}}$ and is a convex combination of $\overrightarrow{\mathcal{W}_{3,4}}$ and $\overrightarrow{\mathcal{W}_{2,1}}$. Similarly, $\overrightarrow{\mathcal{W}'_{3,1}}$ is a convex combination of $\overrightarrow{\mathcal{W}_{3,2}}$ and $\overrightarrow{\mathcal{W}_{2,1}}$. The agreement $\overrightarrow{\mathcal{A}_{3,1}}$ is equal to the agreement P_2 had with P_1 and hence is equal to $\overrightarrow{\mathcal{A}_{2,1}}$. The other agreements of P_3 do not change since no data is exchanged. Since $\overrightarrow{\mathcal{K}'_3} = \overrightarrow{\mathcal{K}'_2}$, $\overrightarrow{\mathcal{W}'_{3,4}}$ and $\overrightarrow{\mathcal{W}'_{3,1}}$ are all inside the same R, P_2 can be eliminated by joining the neighbors of P_2 with P_3. Repeating this elimination for P_1 and P_4 eventually leaves a single peer whose knowledge is equal to $\overrightarrow{\mathcal{G}} \in R$.

The significance of Theorem 3.6.1 is that under the condition described above, P_i can stop sending messages to its neighbors and output $\mathcal{F}\left(\overrightarrow{\mathcal{K}_i}\right)$. If the current state of the system is indeed a termination state, then Theorem 3.6.1 guarantees this is the correct solution. Else, either there must be a message traversing the network, or some peer P_k for whom the condition does not hold. Eventually, P_k will send a message which, once received, may change the knowledge of P_i, and thus guarantee eventual correctness.

Theorem 3.6.1 only discusses about a single region. However, it is fairly easy to extend it to a collection of non-overlapping regions.

Lemma 3.6.2. *Let $C_\omega = \{R_1, R_2, \ldots, R_n\}$ be a set of convex non-overlapping regions in $\mathbb{R}^d$. If for every peer P_i, the region $R_z \in C_\omega$ containing $\overrightarrow{\mathcal{K}_i}$ also contains $\overrightarrow{\mathcal{W}_{i,j}}$ and $\overrightarrow{\mathcal{A}_{i,j}}$ for every neighbor of Γ_i, then for any peer P_k, the knowledge $\overrightarrow{\mathcal{K}_k} \in R_z$.*

Proof. Let $R_z(\vec{x})$ be the region containing a vector $\vec{x}$. Since the state of the system is a termination state, $\mathcal{A}_{i,j} = \mathcal{A}_{j,i}$. Thus, for every two neighbors P_i and P_j, $R\left(\overrightarrow{\mathcal{K}_i}\right) = R_z\left(\overrightarrow{\mathcal{A}_{i,j}}\right) = R_z\left(\overrightarrow{\mathcal{A}_{j,i}}\right) = R_z\left(\overrightarrow{\mathcal{K}_j}\right)$. Assuming the communication graph is connected, this equivalence can be carried over to every peer in the network. □

The result of Lemma 3.6.2, states a condition by which peers can agree on the region which includes $\overrightarrow{\mathcal{G}}$. Each peer chooses R according to its knowledge $\overrightarrow{\mathcal{K}_i}$ and applies the

condition of Theorem 3.6.1 to R. The agreement on the region also implies an agreement on the value of $\mathcal{F}\left(\overrightarrow{\mathcal{G}}\right)$.

Illustration Recall from Section 3.5.5,

- $\overrightarrow{\mathcal{K}_1} = \{(2, 6)\}$
- $\overrightarrow{\mathcal{A}_{1,2}} = \emptyset$
- $\overrightarrow{\mathcal{W}_{1,2}} = \{(2, 6)\}$
- $\overrightarrow{\mathcal{K}_2} = \{(8.5, 10.5)\}$
- $\overrightarrow{\mathcal{A}_{2,1}} = \emptyset$
- $\overrightarrow{\mathcal{W}_{2,1}} = \{(8.5, 10.5)\}$
- $\overrightarrow{\mathcal{G}} = \{(4.6, 7.8)\}$

We may be interested in finding if $\overrightarrow{\mathcal{G}}$ lies inside a circle of radius 10 and center at (0,0) i.e. if $\left|\left|\overrightarrow{\mathcal{G}}\right|\right| = 9.055 < 10$. In this case the output is true. Also, $\left|\left|\overrightarrow{\mathcal{K}_1}\right|\right| = 6.33 < 10$ but $\left|\left|\overrightarrow{\mathcal{K}_2}\right|\right| = 13.51 > 10$. Therefore if the peers decide the output based solely on $\overrightarrow{\mathcal{K}}$, the result will be wrong.

Now let us apply the conditions of Theorem 3.6.1 to the vectors of P_1 and P_2. The region R is the circle of radius 10. For peer P_1:

- $\left|\left|\overrightarrow{\mathcal{K}_1}\right|\right| = 6.33$: **Inside circle**
- $\left|\left|\overrightarrow{\mathcal{A}_{1,2}}\right|\right| = 0$: **Inside circle**
- $\left|\left|\overrightarrow{\mathcal{W}_{1,2}}\right|\right| = 6.33$: **Inside circle**

Therefore the conditions of the theorem dictates that this is a termination state of peer P_1. Similarly for peer P_2:

- $\left|\left|\overrightarrow{\mathcal{K}_2}\right|\right| = 13.51$: **Outside circle**
- $\left|\left|\overrightarrow{\mathcal{A}_{2,1}}\right|\right| = 0$: **Inside circle**
- $\left|\left|\overrightarrow{\mathcal{W}_{2,1}}\right|\right| = 13.51$: **Outside circle**

In this case, however, the conditions of the theorem states that P_2 has not reached a termination state. Therefore P_2 will send messages which will change its state. In Section 3.7.3 we show what message is sent and how each peer outputs the correct result.

We are now in a position to present our local P2P generic algorithm (PeGMA) for computing complex functions in large distributed systems.

3.7 Peer-to-Peer Generic Algorithm (PeGMA) and its Instantiations

This section describes *PeGMA* — a P2P generic local algorithm for computing in large distributed systems. It relies on the results presented in the previous section to compute the value of a given function of the average of the input vectors. As proved below, *PeGMA* is both local and eventually correct. The section proceeds to exemplify how *PeGMA* can be used for solving interesting data mining problems by instantiating it for two different functions: (1) a function that outputs 0 if the norm of the average vector in $\mathbb{R}^d$ is less than ϵ, and 1 otherwise; and (2) a function that outputs 0 inside a Pacman shape in $\mathbb{R}^2$ and 1 outside of it.

Note that for all the algorithms, one of the inputs is the ω-cover. Finding an algorithm which outputs the ω-cover for any function $\mathcal{F}$ is an open research issue. Finite Element technique (FEM) [55] using quadtree (in $\mathbb{R}^2$) or octree (in $\mathbb{R}^3$) can be used to split any arbitrary $\mathcal{F}$ into convex regions. An example is shown in Figure 3.4. However, the cost of building the tree increases exponentially with increase in the dimension of the problem. Randomized algorithms for generating ω-cover offer better complexities as we show in Section 3.7.3.

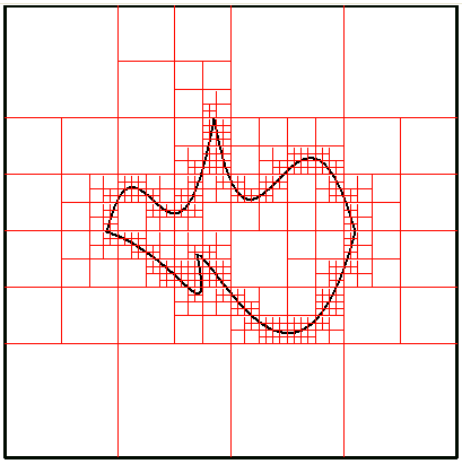


FIG. 3.4. An arbitrary $\mathcal{F}$ is subdivided using quadtree data structure.

3.7.1 PeGMA: Algorithmic Details

PeGMA, depicted in Algorithm 1, receives as input the function $\mathcal{F}$, the ω-cover, which is a set of non-overlapping convex regions, and a constant — L — whose function is explained below. Each peer outputs, at every given time, the value of $\mathcal{F}$ based on its knowledge $\overrightarrow{\mathcal{K}_i}$.

The algorithm is event driven. Events could be one of the following: a message from a neighbor peer, a change in the set of neighbors (e.g., due to failure or recovery), a change in the local data, or the expiry of a timer which is always set to no more than L. On any such event P_i calls the **OnChange** method. If the event is a message $\overrightarrow{X}, |X|$ received from a neighbor P_j, P_i would update $\overrightarrow{X_{i,j}}$ to $\overrightarrow{X}$ and $|X_{i,j}|$ to $|X|$ before it calls **OnChange**.

The objective of the **OnChange** method is to make certain that the conditions of Theorem 3.6.1 are maintained for the peer that runs it. These conditions require $\overrightarrow{\mathcal{K}_i}$, $\overrightarrow{\mathcal{A}_{i,j}}$, and $\overrightarrow{\mathcal{W}_{i,j}}$ (in case it is not null) all to be in the same convex region. Since $\overrightarrow{\mathcal{K}_i}$ cannot be manipulated by the peer, it dictates the active region R. $\overrightarrow{\mathcal{K}_i}$ can also be outside all of the regions in C_ω (when it is in a tie area). Since Theorem 3.6.1 does not provide any guarantee unless

$\overrightarrow{\mathcal{K}_i}$ is in a convex region, eventual correctness can only be guaranteed if the peer floods, in this case, any withheld knowledge it has to its neighbors. Otherwise, the conditions for which data need to be sent are two: (1) if either the agreement $\overrightarrow{\mathcal{A}_{i,j}}$ or the withheld knowledge $\overrightarrow{\mathcal{W}_{i,j}}$ are outside R or (2) if the knowledge $\overrightarrow{\mathcal{K}_i}$ is different from the agreement $\overrightarrow{\mathcal{A}_{i,j}}$, but the size of the withheld knowledge is zero. This can only occur if there is no withheld knowledge at all, and then the data changes.

It should be stressed here that in any case other than those described above, the peer does not need to do anything even if its data changes. The peer can rely on the correctness of the general results from the previous section which ensure that if $\mathcal{F}\left(\overrightarrow{\mathcal{K}_i}\right)$ is not the correct answer then eventually one of its neighbors will send it some new data and change $\overrightarrow{\mathcal{K}_i}$. If, one the other hand, one of the aforementioned cases do occur, then P_i sends a message. This is performed by the **SendMessage** method. If $\overrightarrow{\mathcal{K}_i}$ is in a tie area, then P_i simply sends all of the withheld data. Otherwise, a message is computed which will ensure $\overrightarrow{\mathcal{A}_{i,j}}$ and $\overrightarrow{\mathcal{W}_{i,j}}$ are in R. In *PeGMA* we do not specify exactly how this is done. Nevertheless, one way is evident — sending all of the withheld knowledge would set $\overrightarrow{\mathcal{A}_{i,j}}$ to $\overrightarrow{\mathcal{K}_i}$ (which, from the way we select R, is in R) and $\mathcal{W}_{i,j}$ to zero.

One last mechanism employed in the algorithm is a "leaky bucket" mechanism. This mechanism makes certain that no two messages are sent in a period shorter than a constant L. Leaky bucket is often used in asynchronous, event-based systems to prevent message explosion. Every time a message needs to be sent, the algorithm checks how much time has passed since the last one was sent. If that time is less than L, the algorithm sets a timer for the reminder of the period and calls **OnChange** again when the timer expires. Note that this mechanism does not enforce any kind of synchronization on the system. It also has nothing to do with correctness; at most it might delay convergence. Figure 3.5 shows the flowchart of *PeGMA*.

Input: $\mathcal{F}$, C_ω, L, S_i, and Γ_i
Output: $\mathcal{F}\left(\overrightarrow{\mathcal{K}_i}\right)$

Data structure: For each $P_j \in \Gamma_i$ $\overrightarrow{X_{i,j}}$, $|X_{i,j}|$, $\overrightarrow{X_{j,i}}$, $|X_{i,j}|$, $last\_message$;
if MessageRecvd$\left(P_j, \overrightarrow{X}, |X|\right)$ **then** $\overrightarrow{X_{j,i}} \leftarrow \overrightarrow{X}$ and $|X_{j,i}| \leftarrow |X|$;
if S_i, Γ_i, $\overrightarrow{\mathcal{K}_i}$ *or* $|\mathcal{K}_i|$ *changes* **then** call **OnChange**;
Function ***OnChange***$()$
begin
    Set $R \leftarrow \begin{cases} R_j \in C_\omega & \exists R_j \in C_\omega : \overrightarrow{\mathcal{K}_i} \in R_j \\ \emptyset & Otherwise \end{cases}$
    for *each* $P_j \in \Gamma_i$ **do**
        if $\left[R = \emptyset \bigwedge \left(\overrightarrow{\mathcal{A}_{i,j}} \neq \overrightarrow{\mathcal{K}_i} \bigvee |\mathcal{A}_{i,j}| \neq |\mathcal{K}_i|\right)\right] \bigvee \left[|\mathcal{W}_{i,j}| = 0 \bigwedge \overrightarrow{\mathcal{A}_{i,j}} \neq \overrightarrow{\mathcal{K}_i}\right] \bigvee [\mathcal{A}_{i,j} \notin R \bigvee \mathcal{W}_{i,j} \notin R]$ **then** call **SendMessage**(P_j);
    end
end
Function ***SendMessage***(P_j)
begin
    if $time\,() - last\_message \geq L$ **then**
        Compute new $\overrightarrow{X_{i,j}}$ and $|X_{i,j}|$ such that both $\overrightarrow{\mathcal{K}_i}$, $\overrightarrow{\mathcal{A}_{i,j}}$ and $\overrightarrow{\mathcal{W}_{i,j}}$ are in R;
        if *no such* $\overrightarrow{X_{i,j}}$ *and* $|X_{i,j}|$ *exist* **then**
            $\overrightarrow{X_{i,j}} \leftarrow \frac{|\mathcal{K}_i|\overrightarrow{\mathcal{K}_i} - |X_{j,i}|\overrightarrow{X_{j,i}}}{|\mathcal{K}_i| - |X_{j,i}|}$; $|X_{i,j}| \leftarrow |\mathcal{K}_i| - |X_{j,i}|$;
        end
        $last\_message \leftarrow time\,()$;
        Send $\overrightarrow{X_{i,j}}, |X_{i,j}|$ to P_j;
    end
    else Wait $L - (time\,() - last\_message)$ time units and then call **OnChange**();
end

Algorithm 1: P2P Generic Local Algorithm (*PeGMA*)

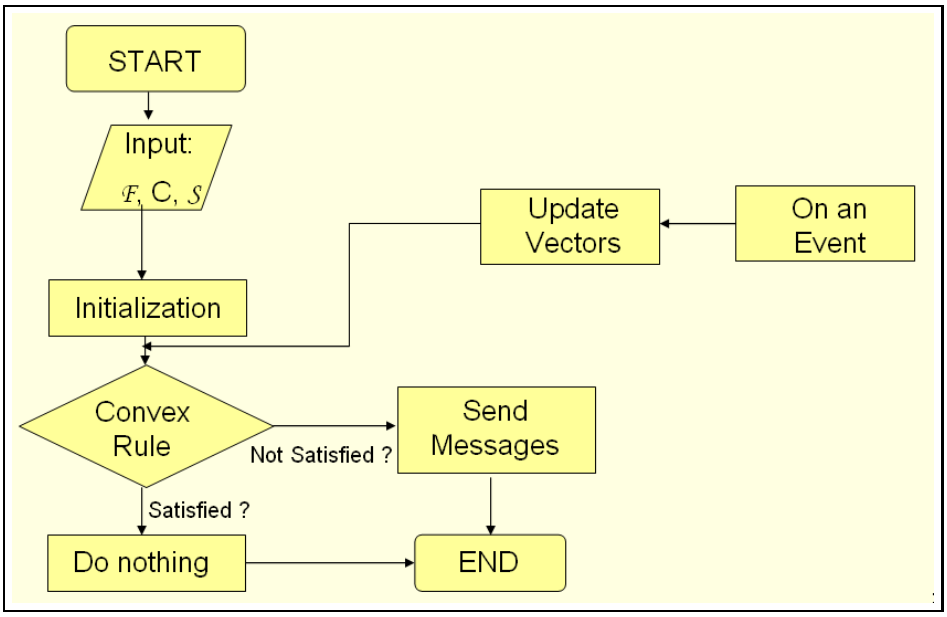


FIG. 3.5. Flowchart of *PeGMA*.

3.7.2 PeGMA: Correctness and Locality

This section proves that *PeGMA* described above is both local and eventually correct.

In order to show eventual correctness we need to prove that if, at some point in time τ, both the underlying communication graph and the data at every peer cease to change, then after some length of time every peer would output the correct result $\mathcal{F}\left(\overrightarrow{\mathcal{G}}\right)$; and that this would happen for *any* static communication tree $G(V, E)$, *any* static data S_i at the peers, and any possible state of the peers at time τ.

Theorem 3.7.1. [Correctness] *The Peer-to-Peer Generic Monitoring Algorithm (PeGMA) is* ***eventually correct****.*

Proof. Regardless of the state of $\mathcal{K}_i$, $\mathcal{A}_{i,j}$, $\mathcal{W}_{i,j}$, the algorithm will continue to send messages, and accumulate more and more of $\mathcal{G}$ in each $\mathcal{K}_i$ until one of two things happen: either for every peer $\mathcal{K}_i = \mathcal{G}$ or for every P_i both $\mathcal{K}_i$, $\mathcal{A}_{i,j}$, and $\mathcal{W}_{i,j}$ are in the same $R_\ell \in C_\omega$. In

the former case, $\overrightarrow{\mathcal{K}_i} = \overrightarrow{\mathcal{G}}$, so every peer obviously computes $\mathcal{F}\left(\overrightarrow{\mathcal{K}_i}\right) = \mathcal{F}\left(\overrightarrow{\mathcal{G}}\right)$. In the latter case, Theorem 3.6.1 dictates that $\overrightarrow{\mathcal{G}} \in R_\ell$, so $\mathcal{F}\left(\overrightarrow{\mathcal{K}_i}\right) = \mathcal{F}\left(\overrightarrow{\mathcal{G}}\right)$ too. □

Lastly, we claim that PeGMA is (α, γ)-local. The art of measuring (α, γ)-locality of algorithms is at its infancy. An attempt has been made to define locality with respect to the *Veracity Radius* of an aggregation problem [15]. However this method does not extend well to algorithms that contains randomness (e.g., in message scheduling) or to dynamic data and topology. Considering the (α, γ) framework we defined earlier, there always exist problem instances for which any eventually correct algorithm (e.g. [116][196][195][119][163] and including all ones described in this dissertation) will have worst case γ = $O(n)$ (as shown in Theorem 3.7.3), where n is the size of the network. While $O(n)$ is the upper bound on the communication complexity, more accurate bounds on γ can be developed by identifying the specific problems and input instances. We feel that there is an intrinsic relation between γ and ϵ. For example increasing ϵ decreases γ though it needs to be investigated further and constitutes part of the future work of this dissertation. Moreover, another factor which actively controls γ is the area of the tie region. As we describe here, this area can be made arbitrarily small by introducing more convex regions. An appropriate measure of accuracy, therefore can be the ratio of the union of the areas spanned by the cover to the tie region.

Lemma 3.7.2. *Considering a two node network, P_i and P_j, the maximum number of messages exchanged between them to come to a consensus about the correct output is 2.*

Proof. Using the notations defined earlier, let $\overrightarrow{\mathcal{K}_i} \in R_k$, $\overrightarrow{\mathcal{K}_j} \in R_\ell$ and $\overrightarrow{\mathcal{G}} \in R_m$, where $R_m, R_k, R_\ell \in C_\omega$ and $m \neq k \neq \ell$. Considering an initialization state, where $\overrightarrow{X_{i,j}} = \overrightarrow{X_{j,i}} = 0$ such that $\overrightarrow{\mathcal{A}_{i,j}} = 0 = \overrightarrow{\mathcal{A}_{j,i}}$. In this case the condition of Theorem 3.6.1 does not hold for both P_i and P_j. P_i will send all of its data (which is $\overrightarrow{\mathcal{K}_i}$) to P_j which will enable P_j to correctly compute $\overrightarrow{\mathcal{G}}$ (since $\overrightarrow{\mathcal{G}}$ is a convex combination of $\overrightarrow{\mathcal{K}_i}$ and $\overrightarrow{\mathcal{K}_j}$). On receiving $\overrightarrow{\mathcal{K}_i}$ from P_i, P_j will apply the conditions of Theorem 3.6.1. Since clearly $\overrightarrow{\mathcal{K}_j} = \overrightarrow{\mathcal{G}} \in R_m$ but

$\overrightarrow{\mathcal{A}}_{j,i} = \overrightarrow{\mathcal{K}_i} \in R_k$, the condition of the theorem dictates it to send a message to P_i and it will send all the data which it has not received from P_i i.e. $\overrightarrow{\mathcal{K}_j}$. At this point both P_i and P_j have both $\overrightarrow{\mathcal{K}_i}$ and $\overrightarrow{\mathcal{K}_j}$. Hence they can compute $\overrightarrow{\mathcal{G}}$ correctly. Therefore the number of messages exchanged is 2.

It can also happen that $\overrightarrow{\mathcal{K}_i}$ or $\overrightarrow{\mathcal{K}_j}$ or both lie inside a tie region. In this case the peers will exchange all of their data and following a similar argument as stated above, we can show that the total number of messages exchanged is 2. □

Our next theorem bounds the total number of messages sent by *PeGMA*. Because of the dependence on the data, counting the number of messages in a data independent manner for such an asynchronous algorithm seems extremely difficult. Therefore in the following theorem (Theorem 3.7.3), we find the upper bound of the number of messages exchanged by any peer when the data of all the peer changes.

Theorem 3.7.3. [Communication Complexity] *Let S_t be a state of the network at time t where for every P_i, $\overrightarrow{\mathcal{K}_i} \in R_\ell$, $R_\ell \in C_\omega$. Hence $\overrightarrow{\mathcal{G}} \in R_\ell$ as well and thus the peers have converged to the correct result. Let at time $t' > t$ the data of each peer changes. Without loss of generality, let us assume that at time t', $\overrightarrow{\mathcal{K}_i} \in R_i$ where each $R_i \in C_\omega$. Let us also assume that $\overrightarrow{\mathcal{G}} \in R_g$, where $g \notin \{1 \dots n\}$. The maximum number of messages sent by any peer P_i is $(n-1) \times (|\Gamma_i| - 1)$ in order to ensure $\overrightarrow{\mathcal{K}_i} \in R_g$.*

Proof. It is clear that the output of each peer will be correct only when each $\overrightarrow{\mathcal{K}_i} = \overrightarrow{\mathcal{G}}$. This will only happen when each P_i has communicated with all the peers in the network i.e. $\overrightarrow{\mathcal{K}_i} = \sum_{i=1}^{n} \overrightarrow{\mathcal{K}_i}$. Since *PeGMA* only communicates with immediate neighbors, in the worst case any peer P_i will be updated with each value of $\overrightarrow{\mathcal{K}_j}$, $j \neq i$ one at a time. Every time P_i gets one $\overrightarrow{\mathcal{K}_j}$, it communicates with all its neighbors except the one from which it got $\overrightarrow{\mathcal{K}_j}$. This process can be repeated in the worst case for $(n-1)$ times in order to get all the $\overrightarrow{\mathcal{K}_j}$'s. At every such update, P_i will communicate with $|\Gamma_i| - 1$ neighbors. Therefore, the total number of messages sent by P_i is $(n-1) \times (|\Gamma_i| - 1)$. □

Our next theorem shows that *PeGMA* is $(O(1), O(n))$-local.

Theorem 3.7.4. [Locality] PeGMA $(O(1), O(n))$-*local.*

Proof. *PeGMA* is designed to work by communicating with immediate neighbors of a peer only. Hence by design, $\alpha = 1$.

From Lemma 3.7.3, we know that $\gamma = O(n)$. Hence, *PeGMA* is $(O(1), O(n))$-local. □

While the worst case $\gamma = O(n)$, there exist many interesting problem instances for which γ is small and independent of the size of the network, as corroborated by our extensive experimental results. Furthermore, the cases for which the communication is global can be controlled to any desired degree by reducing the area of the tie region.

3.7.3 Local L2 Norm Thresholding

Following the description of *PeGMA*, specific algorithms can be implemented for various functions $\mathcal{F}$. One of the most interesting functions is that of thresholding the L2 norm of the average vector, i.e., deciding if $\left\|\overrightarrow{\mathcal{G}}\right\| \leq \epsilon$, where ϵ is a user chosen threshold. Geometrically this means deciding if $\overrightarrow{\mathcal{G}}$ lies inside a circle of radius of ϵ and centered at (0,0).

To produce a specific algorithm from *PeGMA*, the following two steps need to be taken:

1. A ω-cover by non-overlapping convex regions, in each of which $\mathcal{F}_{R_i}$-invariant, needs to be found
2. A method for finding $\overrightarrow{X_{i,j}}$ and $|X_{i,j}|$ which ensures that both $\overrightarrow{\mathcal{A}_{i,j}}$ and $\overrightarrow{\mathcal{W}_{i,j}} \in R$ needs to be formulated

The case of L2 thresholding problem in $\mathbb{R}^2$ is illustrated in Figure 3.6. Obviously, the area for which $\mathcal{F}$ outputs $true$ — the inside of an ϵ-circle — is convex. This area is

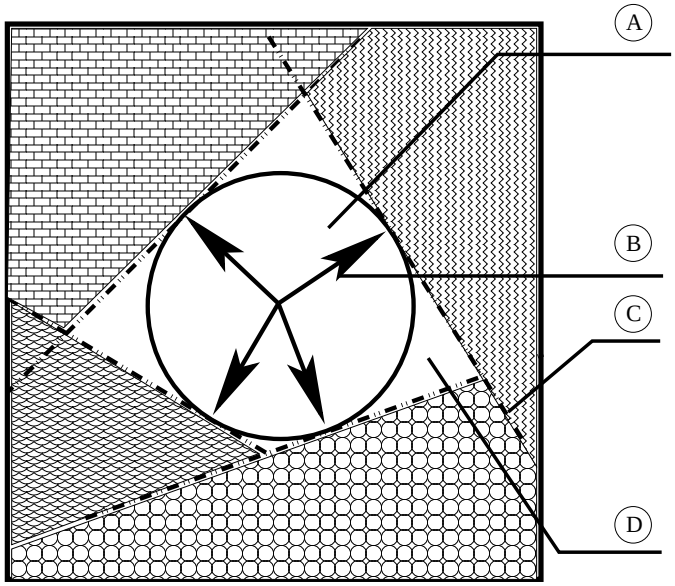


FIG. 3.6. (A) the area inside an ϵ circle (B) A random vector (C) A tangent defining a half-space (D) The areas between the circle and the union of half-spaces are the tie areas

denoted by R_{in}. The area outside the ϵ-circle can be divided by randomly selecting unit vectors $\hat{u_1}, \ldots, \hat{u_\ell}$ and then drawing the half-spaces $H_j = \{\vec{x} : \vec{x} \cdot \hat{u_j} \geq \epsilon\}$. Each half-space is a convex region and is entirely outside the circle. However, these half-spaces overlap. Therefore, non-overlapping regions are defined by removing from each half-space points that belong to the next in order half-space $R_j = \left\{\vec{x} : \vec{x} \cdot \hat{u_j} \geq \epsilon \wedge \vec{x} \cdot u_{(j+1)\hat{} \bmod \ell} < \epsilon\right\}$. By increasing ℓ, the area between the halfspaces and the circle or the tie area can be minimized to any desired degree. Thus, $R_{in}, R_1, \ldots, R_\ell$ constitute a ω-cover. Since the domain can be infinite, we make sure the tie area is less than a ω fraction of the area of the ϵ-circle and thus clearly less than a ω fraction of the entire domain. The pseudo code of the cover for a sphere in R^d is shown in Algorithm 2.

The running time of the algorithm is $O(\ell \times d)$. This is a huge improvement from the exponential time complexity in d using quadtree/octree data structure we discussed earlier.

We now describe how the **SendMessage** method computes a message that forces $\overrightarrow{\mathcal{A}_{i,j}}$

Input: dimension of the problem d, number of unit vectors required ℓ
Output: C_ω
1 Generate ℓ random vectors: $r_1 = \{r_{11}, r_{12}, \ldots, r_{1d}\}$, $r_2 = \{r_{21}, r_{22}, \ldots, r_{2d}\}$, $\ldots$, $r_\ell = \{r_{\ell 1}, r_{\ell 2}, \ldots, r_{\ell d}\}$.
2 Normalize these vectors: $\hat{u_1} = \frac{1}{\sqrt{r_{11}^2 + r_{12}^2 + \cdots + r_{1d}^2}}\{r_{11}, r_{12}, \ldots, r_{1d}\}$
$\hat{u_2} = \frac{1}{\sqrt{r_{21}^2 + r_{22}^2 + \cdots + r_{2d}^2}}\{r_{21}, r_{22}, \ldots, r_{2d}\}$, $\ldots$,
$\hat{u_\ell} = \frac{1}{\sqrt{r_{\ell 1}^2 + r_{\ell 2}^2 + \cdots + r_{\ell d}^2}}\{r_{\ell 1}, r_{\ell 2}, \ldots, r_{\ell d}\}$.
3 $R_{in} = \{\overrightarrow{x} : ||\overrightarrow{x}|| < \epsilon\}$
4 $R_i = \{\overrightarrow{x} : \overrightarrow{x} \cdot \hat{u_i} > \epsilon\}$
5 $C_\omega = \{R_{in}, R_1, \ldots, R_\ell\}$

Algorithm 2: Generic cover for sphere in $\mathbb{R}^d$.

and $\overrightarrow{\mathcal{W}_{i,j}}$ into the region which contains $\overrightarrow{\mathcal{K}_i}$ if they are not in it. A related algorithm, Majority-Rule [196], suggests sending all of the withheld knowledge in this case. However, experiments with dynamic data hint this method may be unfavorable. If all or most of the knowledge is sent and the data later changes, the withheld knowledge becomes the difference between the old and the new data. This difference tends to be far more noisy than the original data. Thus, while the algorithm makes certain that $\overrightarrow{\mathcal{A}_{i,j}}$ and $\overrightarrow{\mathcal{W}_{i,j}}$ are brought into the same region as $\overrightarrow{\mathcal{K}_i}$, it still makes an effort to maintain some withheld knowledge. Though it may be possible to optimize the size of $|\mathcal{W}_{i,j}|$ we take the simple and effective approach of testing an exponentially decreasing sequence of $|\mathcal{W}_{i,j}|$ values, and then choosing the first such value satisfying the requirements for $\overrightarrow{\mathcal{A}_{i,j}}$ and $\overrightarrow{\mathcal{W}_{i,j}}$.

When a peer P_i needs to send a message, it computes $\overrightarrow{X}$, the new value for $\overrightarrow{X_{i,j}}$, based on the contributions of all sources of input vectors other than P_j, $\overrightarrow{X} \leftarrow \frac{|\mathcal{K}_i|\overrightarrow{\mathcal{K}_i} - |X_{j,i}|\overrightarrow{X_{j,i}}}{|\mathcal{K}_i| - |X_{j,i}|}$. Then, it tests a sequence of values for $|X|$, the new weight $|X_{i,j}|$; first setting it to $(|\mathcal{K}_i| - |X_{j,i}|)/2$ and then gradually increasing it until either $\overrightarrow{\mathcal{A}_{i,j}}$ and $\overrightarrow{\mathcal{W}_{i,j}} \in R$ or $|X| = |\mathcal{K}_i| - |X_{j,i}|$.

Input: ϵ, L, S_i, Γ_i, C_ω

Output: 0 if $\left\|\overrightarrow{K_i}\right\| \leq \epsilon$, 1 otherwise

Function *ComputeNew*($|X_{i,j}|$ **and** $\overrightarrow{X_{i,j}}$**)**

begin

  Let R be the region selected by the *PeGMA*;

  $\overrightarrow{X} \leftarrow \frac{|\mathcal{K}_i|\overrightarrow{\mathcal{K}_i} - |X_{j,i}|\overrightarrow{X_{j,i}}}{|\mathcal{K}_i| - |X_{j,i}|}$;

  $w \leftarrow |X| \leftarrow |\mathcal{K}_i| - |X_{j,i}|$;

  repeat

    $w \leftarrow \lfloor \frac{w}{2} \rfloor$;

    $|X| \leftarrow |\mathcal{K}_i| - |X_{j,i}| - w$;

    $\overrightarrow{A} = \frac{|X|\overrightarrow{X} + |X_{j,i}|\overrightarrow{X_{j,i}}}{|X| + |X_{j,i}|}$;

    $|A| = |X| + |X_{j,i}|$;

    $\overrightarrow{B} = \frac{|\mathcal{K}_i|\overrightarrow{\mathcal{K}_i} - |X|\overrightarrow{X} - |X_{j,i}|\overrightarrow{X_{j,i}}}{|\mathcal{K}_i| - |X| - |X_{j,i}|}$;

    $|B| = |\mathcal{K}_i| - |X| - |X_{j,i}|$;

  until $\overrightarrow{A}, \overrightarrow{B} \in R$ *or* $w = 0$;

  Return $|X|$ and $\overrightarrow{X}$ as the new $|X_{i,j}|$ and $\overrightarrow{X_{i,j}}$;

end

Other than these specifications follow the rules specified in *PeGMA*

Algorithm 3: Local L2 Thresholding

Illustration In this section we continue with our example. We want to test if $\overrightarrow{\mathcal{G}}$ lies inside a circle of radius 10. The messages exchanged by each peer is shown in Table 3.1. Note how at the convergence the output of each peer is the correct global output.

3.7.4 Computing Pacman

We now show how *PeGMA* can be suited for another specific function — that of Pacman. The Pacman shape is a circle with one quadrant cut out (See Fig 3.7). Computation of $\mathcal{F}$ inside Pacman is significant since it shows how $\mathcal{F}$ can be computed inside any arbitrary non-convex figure by decomposing the non-convex shape into convex regions.

First we define the ω-cover for this problem.

Let $C_\omega = R_{in}, R_1, \ldots, R_\ell$ be the cover defined in the previous section for the purpose of L2 Thresholding. The cover for the Pacman function is computed by removing R_{in} from the list of covers, and replacing it with three new regions:

1. $R_a = \{\overrightarrow{x} = (r\cos\beta, r\sin\beta) : 0 \leq r \leq \epsilon \wedge 0 \leq \beta < \pi\}$;
2. $R_b = \left\{\overrightarrow{x} = (r\cos\beta, r\sin\beta) : 0 \leq r \leq \epsilon \wedge \pi \leq \beta < \pi + \frac{\pi}{2}\right\}$;
3. $R_c = \left\{\overrightarrow{x} = (r\cos\beta, r\sin\beta) : 0 \leq r \leq \epsilon \wedge \pi + \frac{\pi}{2} \leq \beta < 2\pi\right\}$.

The angle spanned by each of these regions is less than π. Hence each of these regions are convex. The value of $\mathcal{F}$ in each region is constant (1 for the first two, and 0 for the last one). Thus the new cover is a ω-cover. The ω-cover can be written as: $C_\omega = \{R_a, R_b, R_c, R_1, \ldots, R_\ell\}$. In order to monitor $\mathcal{F}$ inside the Pacman, the conditions of Theorem 3.6.1 are applied to each of these regions in C_ω. The output of the Pacman monitoring algorithm would be:

$$\text{Output} = \begin{cases} 0, & \text{if } \overrightarrow{\mathcal{K}_i} \in \{R_c, R_1, \ldots, R_\ell\}; \\ 1, & \text{if } \overrightarrow{\mathcal{K}_i} \in \{R_a, R_b\}; \end{cases}$$

Peer P_1	Peer P_2
Time t_0: $\left\|\left\|\overrightarrow{\mathcal{K}_1}\right\|\right\| = 6.33, \left\|\left\|\overrightarrow{\mathcal{A}_{1,2}}\right\|\right\| = 0, \left\|\left\|\overrightarrow{\mathcal{W}_{1,2}}\right\|\right\| = 6.33$ Output: **true** **All inside circle, so no message**	Time t_0: $\left\|\left\|\overrightarrow{\mathcal{K}_2}\right\|\right\| = 13.51, \left\|\left\|\overrightarrow{\mathcal{A}_{2,1}}\right\|\right\| = 0, \left\|\left\|\overrightarrow{\mathcal{W}_{2,1}}\right\|\right\| = 13.51$ Output: **false** **All not inside circle, so send message** $\|X_{2,1}\|=\|\mathcal{K}_2\| - \|X_{1,2}\| = 2$ $\overrightarrow{X_{2,1}}=\overrightarrow{\mathcal{K}_2} - \overrightarrow{X_{1,2}} = (8.5, 10.5)$ Send $\overrightarrow{X_{2,1}}$, $\|X_{2,1}\|$ to P_1 After message sent: $\left\|\left\|\overrightarrow{\mathcal{K}_2}\right\|\right\| = 13.51, \left\|\left\|\overrightarrow{\mathcal{A}_{2,1}}\right\|\right\| = 13.51, \left\|\left\|\overrightarrow{\mathcal{W}_{2,1}}\right\|\right\| = 0$
Time t_1: **When message received from P_2:** $\|\mathcal{K}_1\|$ = 3 + 2 = 5 $\overrightarrow{\mathcal{K}_1} = \frac{1}{5}\{(6, 18) + (17, 21)\} = \{(4.6, 7.8)\}$ $\left\|\left\|\overrightarrow{\mathcal{K}_1}\right\|\right\|$=9.055 (**Inside**) $\|\mathcal{A}_{1,2}\|$ = 2 $\overrightarrow{\mathcal{A}_{1,2}} = \{(8.5, 10.5)\}$ $\left\|\left\|\overrightarrow{\mathcal{A}_{1,2}}\right\|\right\|$=13.51 (**Outside**) $\|\mathcal{W}_{1,2}\|$ = 5-2 = 3 $\overrightarrow{\mathcal{W}_{1,2}} = \frac{1}{3}\{(23, 39) - (17, 21)\} = \{(2, 6)\}$ $\left\|\left\|\overrightarrow{\mathcal{W}_{1,2}}\right\|\right\|$=6.33 (**Inside**) Output: **true** **All not inside circle, so send message** $\|X_{1,2}\|=\|\mathcal{K}_1\| - \|X_{2,1}\| = 3$ $\overrightarrow{X_{1,2}}=\overrightarrow{\mathcal{K}_1} - \overrightarrow{X_{2,1}} = (2, 6)$ Send $\overrightarrow{X_{1,2}}$, $\|X_{1,2}\|$ to P_2 After message sent: $\left\|\left\|\overrightarrow{\mathcal{K}_1}\right\|\right\| = 9.055, \left\|\left\|\overrightarrow{\mathcal{A}_{1,2}}\right\|\right\| = 9.055, \left\|\left\|\overrightarrow{\mathcal{W}_{1,2}}\right\|\right\| = 0$	
	Time t_2: **When message received from P_1:** $\|\mathcal{K}_2\|$ = 2 + 3 = 5 $\overrightarrow{\mathcal{K}_2} = \frac{1}{5}\{(17, 21) + (6, 18)\} = \{(4.6, 7.8)\}$ $\left\|\left\|\overrightarrow{\mathcal{K}_2}\right\|\right\|$=9.055 (**Inside**) $\|\mathcal{A}_{2,1}\|$ = 5 $\overrightarrow{\mathcal{A}_{2,1}} = \{(4.6, 7.8)\}$ $\left\|\left\|\overrightarrow{\mathcal{A}_{2,1}}\right\|\right\|$=9.055 (**Inside**) $\|\mathcal{W}_{2,1}\|$ = 0 $\overrightarrow{\mathcal{W}_{2,1}} = \overrightarrow{0}$ $\left\|\left\|\overrightarrow{\mathcal{W}_{2,1}}\right\|\right\|$=0 (**Inside**) Output: **true** **All inside circle, so no message**

Table 3.1. Table showing the messages exchanged by two peers P_i and P_j for the L2 thresholding algorithm

.

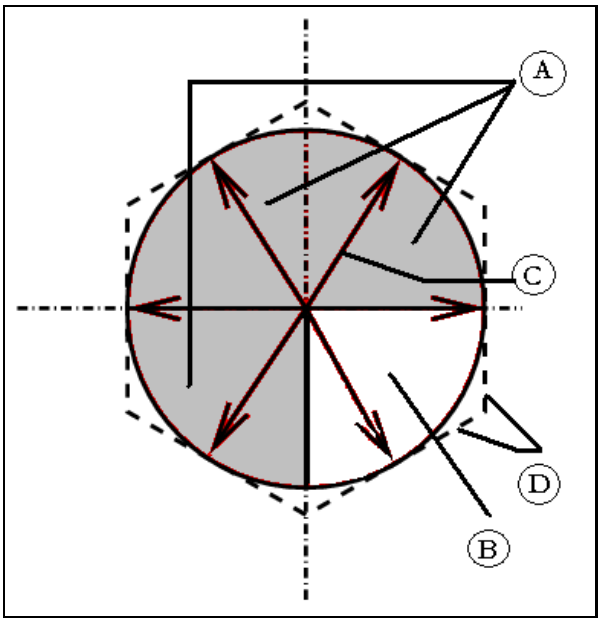


FIG. 3.7. A circle of radius ϵ circumscribing a Pacman shape. (A) Area inside the shape. (B) Area outside the shape. (C) Unit vectors defining the tangent lines. (D) Tangent lines defining the bounding polygon.

The same algorithm employed for L2 Thresholding can be used for the Pacman problem without any additional change.

3.8 Reactive Algorithms

The previous section described *PeGMA* — an efficient local algorithm capable of computing complex functions defined on the average of the data even when the data and system are constantly changing. In this section, we leverage this powerful tool to create a framework for producing and maintaining various data mining models. This framework is simpler than the current methodology of inventing a specific distributed algorithm for each problem.

The basic idea of the framework is to employ a simple, costly, and possibly inaccurate *convergecast* algorithm to sample data from the network to a central post and to compute,

based on this "best-effort" sample, a data mining model. The second part uses a *broadcast* technique to distribute this model in the network. The local algorithm is then used to monitor the quality of the model. If the model is not sufficiently accurate or the data has changed to the degree that the model no longer fits it, the monitoring algorithm alerts and triggers another cycle of data collection. It is also possible to tune the algorithm by increasing the sample size if the alerts are frequent and decreasing it when they are infrequent. Since the monitoring algorithm is eventually correct, eventual convergence to a sufficiently accurate model is very likely. Furthermore, when the data only goes through stationary changes, the monitoring algorithm triggers false alerts infrequently and hence can be extremely efficient. Thus, the overall cost of the framework is low.

Note that the communication complexity of the convergecast process is not local. Rather it is equal to the size of the network. However, this process is only invoked if the data undergoes a change in the underlying distribution. If the number of times this change is low, the total cost is amortized by the stationary periods in which our algorithm is very efficient.

We describe four different instantiations of this basic framework, each highlighting a different aspect. First we discuss the problem of computing the average input vector, to a desired degree of accuracy. Second, we present an algorithm for computing a variant of the k-means clusters suitable for dynamic data. Third, we focus on the problem of computing the first eigenvector of the data. Finally, we discuss the problem of computing multivariate regression in P2P networks.

3.8.1 Mean Monitoring

The problem of monitoring the mean of the input vectors has direct applications to many data analysis tasks. The objective in this problem is to compute a vector $\overrightarrow{\mu}$ which is a good approximation for $\overrightarrow{\mathcal{G}}$. Formally, we require that $\left\| \overrightarrow{\mathcal{G}} - \overrightarrow{\mu} \right\| \leq \epsilon$ for a desired value

of ϵ.

For any given estimate $\overrightarrow{\mu}$, monitoring whether $\left\|\overrightarrow{\mathcal{G}} - \overrightarrow{\mu}\right\| \leq \epsilon$ is possible via direct application of the L2 Thresholding algorithm from Section 3.7.3. Every peer P_i subtracts $\overrightarrow{\mu}$ from every input vector in S_i. Then, the peers jointly execute L2 Norm Thresholding over the modified data. If the resulting average is inside the ϵ-circle then $\overrightarrow{\mu}$ is a sufficiently accurate approximation of $\overrightarrow{\mathcal{G}}$; otherwise, it is not.

The basic idea of the Mean Monitoring algorithm is to employ a convergecast-broadcast process in which the convergecast part computes the average of the input vectors and the broadcast part delivers the new average to all the peers. The trick is that, before a peer sends the data it collected up the convergecast tree, it waits for an indication that the current $\overrightarrow{\mu}$ is not a good approximation of the current data. Thus, when the current $\overrightarrow{\mu}$ is a good approximation, convergecast is slow and only progresses as a result of false alerts. During this time, the cost of the convergecast process is negligible compared to that of the L2 Thresholding algorithm. When, on the other hand, the data does change, all peers alert almost immediately. Thus, convergecast progresses very fast, reaches the root, and initiates the broadcast phase. Hence a new $\overrightarrow{\mu}$ is delivered to every peer which is a more updated estimate of $\overrightarrow{\mathcal{G}}$.

Algorithm Detail The exact algorithm, described in Algorithm 4, adds very little to what is described above. The first addition is that of an alert mitigation constant, τ, selected by the user. The idea here is that an alert should persist for a given period of time before the convergecast advances. Experimental evidence suggests that setting τ to even a fraction of the average edge delay greatly reduces the number of convergecasts without incurring a significant delay in updating $\overrightarrow{\mu}$.

A second detail is the separation of the data used for alerting — the input of the L2 Thresholding algorithm — from that which is used for computing the new average. If the two are the same, then the new average tends to be biased. This is because an alert,

and consequently an advancement in the convergecast, is bound to be more frequent when the local data is extreme. Thus, the initial data, and later on every new data, is randomly associated with one of two datasets: R_i — which is used by the L2 Thresholding algorithm, and T_i on whom the average is computed when convergecast advances.

A third detail is the implementation of the convergecast process. First, every peer tracks changes in the knowledge of the underlying L2 Thresholding algorithm. When it moves from inside the ϵ-circle to outside the ϵ-circle the peer takes note of the time, and sets a timer to τ time units. Whenever a timer expires, or a data message is received from one of its neighbors, P_i first checks if currently there is an alert and if it was recorded τ or more time units ago. If so, it counts from how many of its neighbors it has received data messages. If it has received data messages from all of its neighbors, the peer moves to the broadcast phase, computes the average of its own data and the received data and sends it to itself. If it has received data messages from all but one of the neighbors then this one neighbor becomes the peer's parent in the convergecast tree; the peer computes the average of its own and its other neighbors' data, and sends the average with its cumulative weight to the parent. Then, it moves to the broadcast phase. If it received no data messages from two or more of its neighbors the peer just keeps waiting.

Lastly, the broadcast phase is fairly straightforward. Every peer which receives the new $\overrightarrow{\mu}$ vector, updates its data by subtracting it from every vector in R_i and transfers those vectors to the underlying L2 Thresholding algorithm. Then, if it is in the broadcast phase it sends the new vector to its other neighbors and changes the status to convergecast. There could be one situation in which a peer receives a new $\overrightarrow{\mu}$ vector when it is already in the convergecast phase. This is when two neighbor peers concurrently become roots of the convergecast tree i.e., when each of them concurrently sends the last convergecast message to the other. To break the tie, a root peer P_i which receives $\overrightarrow{\mu}$ from a neighbor P_j while in the convergecast phase compares i to j. If $i > j$ it ignores the message. Otherwise, P_i

treats the message just as it would in a broadcast phase.

Figure 3.8 shows a snap-shot of the convergecast broadcast steps as it progresses up the communication tree. In subfigure 3.8(a), the peers do not raise a flag and only runs the local L2 algorithm. In subfigure 3.8(b), the two leaves raise their flags and send their data up (to the parent) as shown using arrows. Figure 3.8(c) shows an intermediate step where the nodes keep waiting. Finally, the roots (two of them) become activated in subfigure 3.8(d) by exchanging data with each other. The new statistic is computed at both the roots and broadcast. The root with the higher id gets to propagate its result.

3.8.2 k-Means Monitoring

We now turn to a more complex problem, that of computing the k-means of distributed data. The classic formulation of the k-means algorithm is a two step recursive process in which every data point is first associated with the nearest of k centroids, and then every centroid is moved to the average of the points associated with it — until the average is the same as the centroid. To make the algorithm suitable for a dynamic setup, we relax the stopping criteria. In our formulation, a solution is considered admissible when the average of point is within an ϵ-distance of the centroid with whom they are associated.

Similar to the Mean-Monitoring, the k-Means Monitoring algorithm (Algorithm. 7) is performed in a cycle of convergecast and broadcast. The algorithm, however, is different in some important respects. First, instead of taking part of just one execution of L2 Thresholding, each peer takes part in k such executions — one per centroid. The input of the ℓ^{th} execution are those points in the local data set S_i for which the ℓ^{th} centroid, $\overrightarrow{c_\ell}$, is the closest. Thus, each execution monitors whether one of the centroids needs to be updated. If even one execution discovers that the norm of the corresponding knowledge $\left\|\overrightarrow{\mathcal{K}_i^\ell}\right\|$ is greater than ϵ the peer raises an alert. If the alert persists for τ time units, the peer advances to the convergecast process.

Input: ϵ, L, S_i, Γ_i, $\overrightarrow{\mu_0}$, τ
Output: $\overrightarrow{\mu}$ such that $\left\| \overrightarrow{\mathcal{G}} - \overrightarrow{\mu} \right\| < \epsilon$

Data structure: R_i and T_i, $last\_change$, flags: $alert$, $root$, and $phase$, for each $P_j \in \Gamma_i$, a vector $\overrightarrow{v}_j$ and a counter c_j;
Initialization:
begin
    Set $\overrightarrow{\mu} \leftarrow \overrightarrow{\mu_0}$, $alert \leftarrow false$, $phase \leftarrow convergecast$;
    Split S_i evenly between R_i and T_i;
    Initialize L2 with the input $\left\{ \overrightarrow{x_{i,1}} - \overrightarrow{\mu}, \overrightarrow{x_{i,2}} - \overrightarrow{\mu}, \ldots, \overrightarrow{x_{i,B}} - \overrightarrow{\mu} : \overrightarrow{x_{i,j}} \in R_i \right\}$;
    Set $\overrightarrow{v_j}, c_j$ to $\overrightarrow{0}, 0$ for all $P_j \in \Gamma_i$;
end
On addition of a new vector $\overrightarrow{x}$ to S_i:
begin
    Randomly add $\overrightarrow{x}$ to either R_i or T_i and remove the oldest point;
    if $\overrightarrow{x}$ *was added to* R_i **then** pass $\overrightarrow{x} - \overrightarrow{\mu}$ to the L2 thresholding algorithm;
end
On change in $\overrightarrow{\mathcal{K}_i}$ of the L2 thresholding algorithm:
begin
    if $\left\| \overrightarrow{\mathcal{K}_i} \right\| \geq \epsilon$ *and* $alert = false$ **then**
        $last\_change \leftarrow time()$;
        $alert \leftarrow true$;
        timer $\leftarrow \tau$;
    end
    if $\left\| \overrightarrow{\mathcal{K}_i} \right\| < \epsilon$ **then** $alert \leftarrow false$;
end
if DataReceived $(P_j, \overrightarrow{v}, c)$ **then**
    $\overrightarrow{v_j} \leftarrow \overrightarrow{v}$, $c_j \leftarrow c$;
    Call **Convergecast** (See Alg. 5 for pseudocode);
end
if *timer expires* **then** Call **Convergecast** (See Alg. 5 for pseudocode);
if *new* μ' *received* **then** Call **ProcessNewMeans(μ')** (See Alg. 6 for pseudocode);

Algorithm 4: Mean Monitoring.

Function ***Convergecast***
begin
  if $alert = false$ **then** return;
  if $time() - last\_change < \tau$ **then**
    $timer \leftarrow time() + \tau - last\_change$;
    return;
  end
  if $\forall P_k \in \Gamma_i$ *except for one* $c_k \neq 0$ **then**
    $s = |T_i| + \sum_{P_j \in \Gamma_i} c_j$;
    $\overrightarrow{s} = \frac{|T_i|}{s}\overrightarrow{T_i} + \sum_{P_j \in \Gamma_i} \frac{c_j}{s}\overrightarrow{v_j}$;
    Send $s, \overrightarrow{s}$ to P_l;
    Set $root \leftarrow false$, $phase \leftarrow Broadcast$;
  end
  if $\forall P_k \in \Gamma_i\ c_k \neq 0$ **then**
    $s = |T_i| + \sum_{P_j \in \Gamma_i} c_j$;
    $\overrightarrow{s} = \frac{|T_i|}{s}\overrightarrow{T_i} + \sum_{P_j \in \Gamma_i} \frac{c_j}{s}\overrightarrow{v_j}$;
    $root \leftarrow true$;
    $phase \leftarrow Convergecast$;
    Send $\overrightarrow{\mu}$ to all $P_k \in \Gamma_i$
  end
end

Algorithm 5: Convergecast function for Mean Monitoring.

Function ***ProcessNewMeans***$(\overrightarrow{\mu'})$
begin
  if $(phase = broadcast \bigvee phase = convergecast) \bigwedge root = true \bigwedge j > i$ **then**
    $\overrightarrow{\mu} \leftarrow \overrightarrow{\mu'}$;
    Send $\overrightarrow{\mu}$ to all $P_k \neq P_j \in \Gamma_i$;
    Replace S_i of the L2 Thresholding algorithm with $\left\{\overrightarrow{x} - \overrightarrow{\mu} : \overrightarrow{x} \in R_i\right\}$;
    $phase \leftarrow convergecast$;
  end
end

Algorithm 6: Function handling the receipt of new means μ for Mean Monitoring.

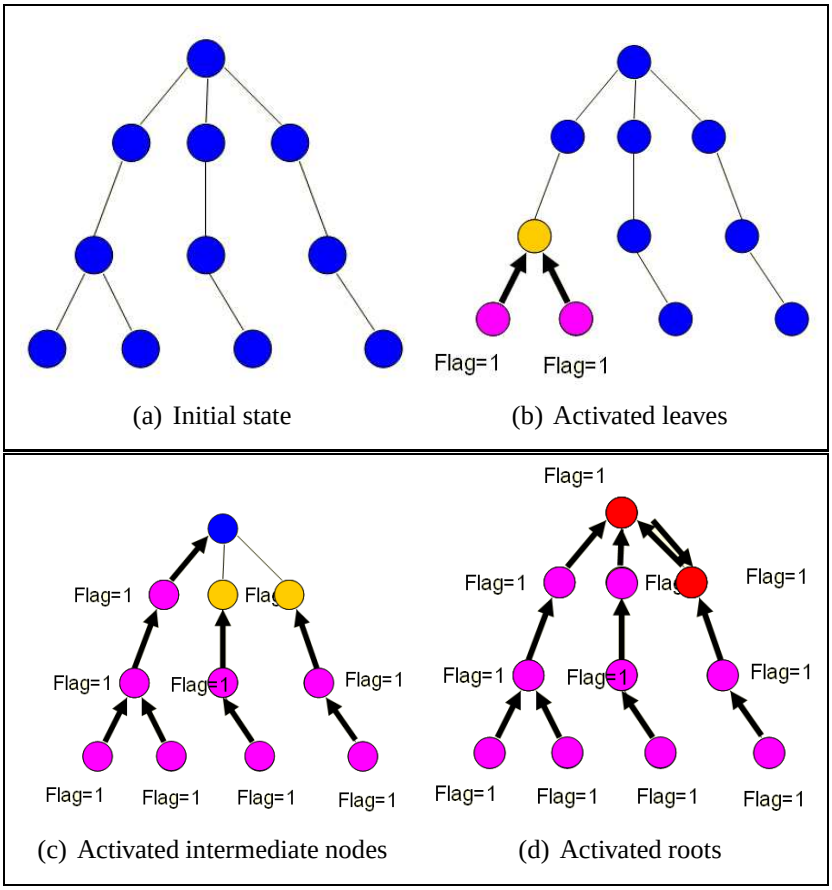


FIG. 3.8. Convergecast and broadcast through the different steps. In subfigure 3.8(a), the peers do not raise a flag. In subfigure 3.8(b), the two leaves raise their flags and send their data up (to the parent) as shown using arrows. Figure 3.8(c) shows an intermediate step. Finally, the roots (two of them) become activated in subfigure 3.8(d) by exchanging data with each other.

Another difference between the k-Means Monitoring and the Mean-Monitoring algorithm is the statistics collected during convergecast. In k-Means Monitoring, the statistics is a sample of size b (dictated by the user) from the data. Each peer samples (with returns) from the samples it received from its neighbors, and from its own data, such that the probability of sampling a point is proportional to the weight. The result of this procedure is that every input point stands an equal chance to be included in the sample that arrives to the root. The root then computes a centralized k-Means on the sample, and sends the new centroids in a broadcast message.

3.8.3 Eigenvector Monitoring

The next algorithm described in this chapter is an algorithm for monitoring the first eigenvector of data. In this problem we assume that the data at each peer, S_i, is a matrix. The objective of the algorithm is to compute the first eigenvector and the corresponding eigenvalue of the global data — the point-wise addition of all of the local matrices. For this purpose, the algorithm takes advantage of the accepted Power Method. The Power Method is a numeric method for approximating the first eigenvector — eigenvalue pair of a matrix $\mathcal{M}$. It begins with a random guess $\overrightarrow{V}$ and θ of the eigenvector-eigenvalue pair and keeps updating them until $\left\|\mathcal{M}\overrightarrow{V} - \theta\overrightarrow{V}\right\| \leq \epsilon$.

Given the formulation above, the computational task of eigenvector monitoring reverts to exactly L2 Thresholding. For a given guess $\overrightarrow{V}$, θ, the peers need to compute if

$$\begin{aligned} &\left\|[\textstyle\sum_i S_i] \cdot \overrightarrow{V} - \theta\overrightarrow{V}\right\| \leq \epsilon \\ \Longrightarrow &\left\|\textstyle\sum_i \left[S_i \cdot \overrightarrow{V}\right] - \theta\overrightarrow{V}\right\| \leq \epsilon \\ \Longrightarrow &\left\|\tfrac{1}{n}\textstyle\sum_i \left[S_i \cdot \overrightarrow{V}\right] - \tfrac{\theta}{n}\overrightarrow{V}\right\| \leq \tfrac{\epsilon}{n} \end{aligned}$$

For every P_i, $\left[S_i \cdot \overrightarrow{V}\right]$ can be computed locally and $\frac{1}{n}\sum_i \left[S_i \cdot \overrightarrow{V}\right]$ is simply the average of those vectors. On the other hand, $\overrightarrow{V}$ and $\frac{\theta}{n}\overrightarrow{V}$ are constants computed and distributed

Input: ϵ, L, S_i, Γ_i, an initial guess for the centroids C_0, τ, b
Output: k centroids such that the average of the points assigned to every centroid is within ϵ of that centroid.

Data structure of peer P_i: A partitioning of S_i into k sets $S_i^1 \dots S_i^k$, $C = \left\{\overrightarrow{c_1}, \dots, \overrightarrow{c_k}\right\}$, for each centroid $j = 1, \dots, k$, a flag $alert_j$, a timestamp $last\_change_j$, a dataset B_j and a counter b_j, a flag $root$ and a flag $phase$.
Initialization:
begin
    $C \leftarrow C_0$;
    for $j = 1 \dots k$ **do** $S_i^j = \left\{\overrightarrow{x} \in S_i : \overrightarrow{c_j} = \arg\min_{\ell=1\dots k} \left\|\overrightarrow{x} - \overrightarrow{c_\ell}\right\|\right\}$;
    Initialize k instances of L2 thresholding algorithm, such that the j^{th} instance has input data $\left\{\overrightarrow{x} - \overrightarrow{c_j} : \overrightarrow{x} \in S_i^j\right\}$;
    forall $P_j \in \Gamma_i$ **do** $b_j \leftarrow 0$;
    for $j = 1, \dots, k$ **do**
        $alert_j \leftarrow false$;
        $last\_change_j \leftarrow -\infty$;
        $root \leftarrow false$;
        $phase \leftarrow convergecast$;
    end
end
On addition of a new vector $\overrightarrow{x}$ to S_i:
begin
    Find c_j closest to $\overrightarrow{x}$ and add $\overrightarrow{x} - \overrightarrow{c_j}$ to j^{th} L2 instance;
end
On removal of a vector $\overrightarrow{x}$ from S_i:
begin
    Find c_j closest to $\overrightarrow{x}$ and remove $\overrightarrow{x} - \overrightarrow{c_j}$ from j^{th} L2 instance;
end
On change in $\overrightarrow{\mathcal{K}_i}$ of the j^{th} instance of the L2 thresholding algorithm:
begin
    if $\left\|\overrightarrow{\mathcal{K}_i}\right\| \geq \epsilon$ *and* $alert_j = false$ **then**
        $last\_change_j \leftarrow time()$;
        $alert_j \leftarrow true$;
        $timer \leftarrow \tau$ time units;
    end
    if $\left\|\overrightarrow{\mathcal{K}_i}\right\| < \epsilon$ **then** $alert_j \leftarrow false$;
end
if DataReceived*(P_j, B, b)* **then**
    $B_j \leftarrow B$, $b_j \leftarrow b$;
    call **Convergecast** (See Algo. 8 for pseudocode);
end
if *timer expires* **then** Call **Convergecast** (See Alg. 8 for pseudocode);
if *new C' received* **then** Call **ProcessNewCentroids(C')** (See Alg. 9 for pseudocode);

Algorithm 7: P2P k-Means Clustering

```
1  Function Convergecast
2  begin
3      if for ℓ ∈ [1, ..., k] then alert_ℓ = false then return;
4      t ← Min_{ℓ=1...k} {last_message_ℓ : alert_ℓ = true};
5      A be a set of b samples returned by Sample;
6      if time() < t + τ then
7          timer ← t + τ − time();
8          return;
9      end
10     if ∀P_k ∈ Γ_i except for one b_k ≠ 0 then
11         root ← false;
12         phase ← Broadcast;
13         Send A, 1 + Σ_{m=1...} b_m to P_ℓ;
14         return;
15     end
16     if ∀P_k ∈ Γ_i b_k ≠ 0 then
17         C′ ← centroids by computing the k-means clustering of A;
18         root ← true;
19         Send C′ to self;
20         return;
21     end
22 end
23 Function Sample
24 begin
25     Return a random sample from S_i with probability 1/(1 + Σ_{m=1...|Γ_i|} b_m) or
       from a dataset B_j with probability b_j/(1 + Σ_{m=1...|Γ_i|} b_m);
26 end
```

Algorithm 8: Convergecast function for P2P k-Mean Monitoring.

```
1  Function ProcessNewCentroids(C⃗′)
2  begin
3      if (phase = Broadcast ⋁ phase = convergecast) ⋀ root = true ⋀ j > i
       then
4          C ← C′;
5          for j = 1 . . . k do S_i^j ← { x⃗ ∈ S_i : c_j⃗ = arg min_{ℓ=1...k} ‖x⃗ − c_ℓ⃗‖ };
6          for j = 1 . . . |Γ_i| do b_j ← 0;
7          Restart all the L2 Thresholding instances with the new centroid;
8          Send C to all P_k ≠ P_j ∈ Γ_i;
9          phase ← Convergecast;
10     end
11 end
```

Algorithm 9: Function handling the receipt of new centroids C' for P2P k-means Monitoring.

using broadcast. The problem is therefore equivalent to Mean-Monitoring with $S_i \cdot \overrightarrow{V}$ taking the place of $\overrightarrow{S_i}$, $\frac{\theta}{n}\overrightarrow{V}$ the place of $\overrightarrow{\mu}$, and $\frac{\epsilon}{n}$ the place of ϵ.

The rest is very similar to k-Means Monitoring. The peers monitor the average vector for the current guess of $\overrightarrow{V}$ and θ. If the resulting vector is greater than $\frac{\epsilon}{n}$, they advance the convergecast process which samples the data to a root. The root then uses the sampled data to compute new $\overrightarrow{V}$ and θ, that are broadcast to all of the peers. Because of the similarity of the algorithm to the k-Means Monitoring, we avoid presenting the pseudo code here.

3.8.4 Distributed Multivariate Regression

The last algorithm discussed in this chapter is a distributed multivariate regression (MR) algorithm. Since this algorithm is different in many aspects compared to the other algorithms presented in this chapter, we start with some background material before we describe the algorithm.

P2P multivariate regression considers building and updating regression models from data distributed over a P2P network where each peer contains a subset of the data tuples.

In the distributed data mining literature, this is usually called the horizontally partitioned or homogeneously distributed data scenario. Building a global regression model (defined on the union of all the data of all the peers) in large-scale networks and maintaining it is a vital task. Consider a network where there are a number of nodes and each node gets a stream of tuples (can be sensor readings, music files etc.) every few seconds thereby generating huge volume of data. We may wish to build a regression model on the global data to (1) compactly represent the data and (2) predict the value of a target variable. This is difficult since the data is distributed and more so because it is dynamic. Centralization obviously does not work because the data may change at a faster rate than the rate at which it can be centralized and it can be too costly. Local algorithms are an excellent choice in such scenarios since in a local algorithm, each peer computes the result based on the information from only a handful of nearby neighbors. Hence local algorithms are highly scalable. Furthermore, they guarantee that once the computation terminates, each node will have the correct regression model. Therefore, such an algorithm will enable the user to monitor regression models using low resources and high accuracy.

Related Work The problem of distributed multivariate regression has been addressed by many researchers till date. Hershberger *et al.* [78] considered the problem of performing global MR in a vertically partitioned data distribution scenario. The authors propose a wavelet transform of the data such that, after the transformation, effect of the cross terms can be dealt with easily. The local MR models are then transported to the central site and combined to form the global MR model. Such synchronized techniques will not scale in large, asynchronous systems such as modern P2P networks.

Many researchers have looked into the problem of doing distributed MR using distributed kernel regression techniques such as Guestrin *et al.* [72] and Predd *et al.* [153]. The algorithm presented by Guestrin *et al.* [72] performs linear regression in a network of sensors using in-network processing of messages. Instead of transmitting the raw data, the

proposed technique transmits constraints only, thereby reducing the communication complexity drastically. Similar to the work proposed here, their work also uses local rules to prune messages. However the major drawback is that their algorithm is not suitable for dynamic data. It will be very costly if the data changes since, as the authors point out, that two passes are required over the entire network to make sure that the effect of the measurements of each node are propagated to every other node. Moreover, contrary to the broad class of problems that we can solve using our technique, their technique is only applicable for solving the linear regression problem.

Linear Regression from heterogeneously distributed data using variational approximation technique has been proposed by Mukherjee and Kargupta [131]. In the paper, the authors pose the original linear regression problem as a convex optimization problem and then use variation approximation technique to solve it. Distributed computation of the regression coefficients reduces to distributed inner product computation and the authors use the technique proposed by Egecioglu *et al.* [143] to compute the inner products efficiently. This distributed regression technique is scalable as corroborated by the theoretical analysis and experimental results.

Meta-learning is an interesting class of algorithms typically used for supervised learning. In a meta learning, such as bagging [21] or boosting [62] many models are induced from different partitions of the data and these "weak" models are combined using a second level algorithm which can be as simple as taking the average output of the models for any new sample. Such a technique is suitable for inducing models from distributed data as proposed by Stolfo *et al.* [176]. The basic idea is to learn a model at each site locally (no communication at all) and then, when a new sample comes, predict the output by simply taking an average of the local outputs. Xing *et al.* [197] present such a framework for doing regression in heterogenous datasets. However, these techniques require synchronization since the locally learned models are shipped to a central site for the final combination.

Notation and Problem Definition The local data of peer P_i at time t is a set of vectors in $\mathbb{R}^d$ denoted by S_i=$\left[\left(\overrightarrow{x_1^i}, f(\overrightarrow{x_1^i})\right), \left(\overrightarrow{x_2^i}, f(\overrightarrow{x_2^i})\right), \ldots, \left(\overrightarrow{x_s^i}, f(\overrightarrow{x_s^i})\right)\right]$. Each d-dimensional data vector consists of two components — the *input* set of attributes $\overrightarrow{x_j^i}$=$\left[x_{j1}^i x_{j2}^i \ldots x_{j(d-1)}^i\right]$ of dimensionality $\mathbb{R}^{d-1}$ and the real-valued *output* $f\left(\overrightarrow{x_j^i}\right)$. Therefore each data vector in S_i can be viewed as an input-output pair. The function f is defined as follows: $f : \mathbb{R}^{d-1} \rightarrow \mathbb{R}$. The rest of the notations are the ones defined in Section 3.5.1.

In MR, the task is to learn a function $\widehat{f}(\overrightarrow{x})$ which "best" approximates $f(\overrightarrow{x})$ according to some measure such as least square. Now depending on the representation chosen for $\widehat{f}(\overrightarrow{x})$, various types of regression models (linear or nonlinear) can be developed. We leave this type specification as part of the problem statement for our algorithm, rather than an assumption.

In MR, for each data point $\overrightarrow{x}$, the error between $\widehat{f}(\overrightarrow{x})$ and $f(\overrightarrow{x})$ can be computed as $\left[f(\overrightarrow{x}) - \widehat{f}(\overrightarrow{x})\right]^2$. In our scenario, since this error value is distributed across the peers, a good estimate of the global error is the average error across all the peers, denoted as $Avg\left[f(\overrightarrow{x}) - \widehat{f}(\overrightarrow{x})\right]^2$. There exist several methods for measuring how suitable a model is for the data under consideration. We have used the L2-norm distance between the current network data and the model as a quality metric for the model built. Given a dynamic environment, our goal is to maintain a $\widehat{f}(\overrightarrow{x})$ at each peer at any time which best approximates $f(\overrightarrow{x})$.

Problem 1. [**MR Problem**] Given a time varying dataset S_i, a user-defined threshold ϵ and $\widehat{f}(\overrightarrow{x})$ to all the peers, the MR problem is to maintain a $\widehat{f}(\overrightarrow{x})$ at each peer such that, at any time t, $\left|\left|Avg\left[f(\overrightarrow{x}) - \widehat{f}(\overrightarrow{x})\right]^2\right|\right| < \epsilon$.

For ease of explanation, we decompose this task into two subtasks. First, given $\widehat{f}(\overrightarrow{x})$ to all the peers, we want to raise an alarm whenever $\left|\left|Avg\left[f(\overrightarrow{x}) - \widehat{f}(\overrightarrow{x})\right]^2\right|\right| > \epsilon$, where

ϵ is a user-defined threshold. This is the *model monitoring problem*. Secondly, if $\widehat{f}(\overrightarrow{x})$ no longer represents $f(\overrightarrow{x})$, we sample from the network (or even better do an in-network aggregation) to find an updated $\widehat{f}(\overrightarrow{x})$. This is the *model computation problem*. Mathematically, the subproblems can be formalized as follows.

Problem 2.[**Monitoring Problem**] Given S_i, and $\widehat{f}(\overrightarrow{x})$ to all the peers, the monitoring problem is to output 0 if $\left|\left| Avg\left[f(\overrightarrow{x}) - \widehat{f}(\overrightarrow{x})\right]^2 \right|\right| < \epsilon$, and 1 otherwise, at any time t.

Problem 3.[**Computation Problem**] The model computation problem is to find a new $\widehat{f}(\overrightarrow{x})$ based on a sample of the data collected from the network.

Approach With the two problems defined earlier, it is easy to see that the monitoring problem reverts exactly to the L2 Monitoring algorithm. The global average error between the true model and the computed one is a point in $\mathbb{R}$. In order to use L2 norm thresholding as the metric for tracking this average error, we transform this 1-D problem to a 2-D problem by defining a vector in $\mathbb{R}^2$ with the first component set to the average error for the peer and the second component set to 0. Therefore, determining if the average error is less than ϵ is equivalent to finding if the average error vector lies inside a circle of radius ϵ, i.e. tracking the L2-norm of the global average error.

The second problem can be solved using the application of a convergecast-broadcast technique. Once the L2 thresholding algorithm raises a flag, samples are collected and propagated up the tree using a convergecast process, the new model $\widehat{f}(\overrightarrow{x})$ is computed at the root and then propagated down the tree (the broadcast process). One interesting point to note is that if $f(\overrightarrow{x})$ is linear in terms of the weights of the regression model, the convergecast process is exact rather than approximate.

Example In this section we illustrate the P2P MR algorithm. Let there be two peers P_i and P_j. Let the regression model be linear in the regression coefficients: $a_0+a_1x_1+a_2x_2$, where a_0, a_1 and a_2 are the regression coefficients having values 1, 2 and -2 respectively and x_1 and x_2 are the two attributes of the data. The coefficients are given to all the peers. The data of peer P_i is $S_i = \{(3, 1, 3.9), (0, -1, 3.6)\}$, where the third entry of each data point is the output generated according to the regression model. To this, we add 30% noise. Similarly, for peer P_j, $S_j = \{(1, 4, -6.5), (-3, 2, -9.1)\}$. Now for peer P_i, the squared error for each point is: $\{(0.9)^2, (0.6)^2\}$. Similarly for P_j, the errors are $\{(1.5)^2, (2.1)^2\}$. The average error is $\frac{(0.9)^2+(0.6)^2+(1.5)^2+(2.1)^2}{4} = 1.9575$. In $\mathbb{R}^2$, the task is to determine if $||1.9575, 0|| > \epsilon$ i.e. $3.83 > \epsilon$, for a user defined ϵ.

Special case : Linear Regression In many cases, sampling from the network is communication intensive. We can find the coefficients using an in-network aggregation if we choose to monitor a widely used regression model *viz.* linear regression (linear with respect to the parameters or the unknown weights).

Let the global dataset over all the peers be denoted by:

$$\mathcal{G} = \begin{pmatrix} x_{11} & x_{12} & \dots & x_{1(d-1)} & f(\overrightarrow{x_1}) \\ x_{21} & x_{22} & \dots & x_{2(d-1)} & f(\overrightarrow{x_2}) \\ \vdots & \vdots & & \vdots & \vdots \\ x_{j1} & x_{j2} & \dots & x_{j(d-1)} & f(\overrightarrow{x_j}) \\ \vdots & \vdots & & \vdots & \vdots \\ x_{|\mathcal{G}|1} & x_{|\mathcal{G}|2} & \dots & x_{|\mathcal{G}|(d-1)} & f(\overrightarrow{x_{|\mathcal{G}|}}) \end{pmatrix}$$

where $\overrightarrow{x_j} = \{x_{j1}x_{j2}\dots x_{j(d-1)}\}$.

In MR, the idea is to learn a function $\widehat{f}(\overrightarrow{x_j})$ which approximates $f(\overrightarrow{x_j})$ for all the data points in $\mathcal{G}$. For linear regression, that function $\widehat{f}(\overrightarrow{x_j})$ is chosen to be a linear function i.e.

a $d-1$ degree polynomial fitted to the input attribute points $\left\{x_{j1}x_{j2}\ldots x_{j(d-1)}\right\}\ \forall j = 1$ to $|\mathcal{G}|$. More specifically, the linear model which we want to fit be: $\widehat{f}(\overrightarrow{x_j}) = a_0 + a_1x_{j1} + a_2x_{j2} + ... + a_{j(d-1)}x_{d-1}$, where a_i's are the coefficients that need to be estimated from the global dataset $\mathcal{G}$. We drop the cross terms involving x_{jk} and $x_{j\ell}$ for simplicity $\forall k, \ell \in [1..(d-1)]$.

For every data point in the set $\mathcal{G}$, the squared error is:

$$\begin{aligned}
E_1 &= \left[f\left(\overrightarrow{x_1}\right) - a_0 - a_1x_{11} - a_2x_{12} - \ldots - a_{d-1}x_{1(d-1)}\right]^2 \\
E_2 &= \left[f\left(\overrightarrow{x_1}\right) - a_0 - a_1x_{21} - a_2x_{22} - \ldots - a_{d-1}x_{2(d-1)}\right]^2 \\
&\vdots \\
E_{|\mathcal{G}|} &= \left[f\left(\overrightarrow{x_{|\mathcal{G}|}}\right) - a_0 - a_1x_{|\mathcal{G}|1} - a_2x_{|\mathcal{G}|2} - \ldots - a_{d-1}x_{|\mathcal{G}|(d-1)}\right]^2
\end{aligned}$$

Thus the total square error over all the data points is

$$SSE = \sum_{j=1}^{|\mathcal{G}|} E_j = \sum_{j=1}^{|\mathcal{G}|} \left[f\left(\overrightarrow{x_j}\right) - a_0 - a_1x_{j1} - a_2x_{j2} - \ldots - a_{d-1}x_{j(d-1)}\right]^2$$

For linear regression, closed form expressions exist for finding the coefficients a_i's by finding the partial derivatives of SSE with respect to the a_i's and setting them to zero:

$$
\begin{aligned}
\frac{\partial}{\partial a_0} SSE &= 2\sum_{j=1}^{|\mathcal{G}|}\left[f\left(\overrightarrow{x_j}\right) - a_0 - a_1 x_{j1} - a_2 x_{j2} - \ldots - a_{d-1}x_{j(d-1)}\right](-1) = 0 \\
\frac{\partial}{\partial a_1} SSE &= 2\sum_{j=1}^{|\mathcal{G}|}\left[f\left(\overrightarrow{x_j}\right) - a_0 - a_1 x_{j1} - a_2 x_{j2} - \ldots - a_{d-1}x_{j(d-1)}\right](-x_{j1}) = 0 \\
\frac{\partial}{\partial a_2} SSE &= 2\sum_{j=1}^{|\mathcal{G}|}\left[f\left(\overrightarrow{x_j}\right) - a_0 - a_1 x_{j1} - a_2 x_{j2} - \ldots - a_{d-1}x_{j(d-1)}\right](-x_{j2}) = 0 \\
&\vdots \\
\frac{\partial}{\partial a_{d-1}} SSE &= 2\sum_{j=1}^{|\mathcal{G}|}\left[f\left(\overrightarrow{x_j}\right) - a_0 - a_1 x_{j1} - a_2 x_{j2} - \ldots - a_{d-1}x_{j(d-1)}\right](-x_{j(d-1)}) = 0
\end{aligned}
$$

In the matrix form this can be written as:

$$
\begin{pmatrix}
|\mathcal{G}| & \sum_{j=1}^{|\mathcal{G}|} x_{j1} & \ldots & \sum_{j=1}^{|\mathcal{G}|} x_{j(d-1)} \\
\sum_{j=1}^{|\mathcal{G}|} x_{j1} & \sum_{j=1}^{|\mathcal{G}|} (x_{j1})^2 & \ldots & \sum_{j=1}^{|\mathcal{G}|} x_{j1} * x_{j(d-1)} \\
\vdots & \vdots & \ddots & \vdots \\
\sum_{j=1}^{|\mathcal{G}|} x_{j(d-1)} & \sum_{j=1}^{|\mathcal{G}|} x_{j(d-1)} * x_{j1} & \ldots & \sum_{j=1}^{|\mathcal{G}|} (x_{j(d-1)})^2
\end{pmatrix}
\times
\begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_{d-1} \end{pmatrix}
=
\begin{pmatrix}
\sum_{j=1}^{|\mathcal{G}|} f(\overrightarrow{x_j}) \\
\sum_{j=1}^{|\mathcal{G}|} f(\overrightarrow{x}_j) x_{j1} \\
\vdots \\
\sum_{j=1}^{|\mathcal{G}|} f(\overrightarrow{x}_j) x_{j(d-1)}
\end{pmatrix}
\Rightarrow \mathbf{Xa} = \mathbf{Y}
$$

Therefore for computing the matrix (or more appropriately vector) **a**, we need to evaluate the matrices **X** and **Y**. This can be done in a communication efficient manner by noticing that the entries of these matrices are simply sums. Consider the distributed scenario where $\mathcal{G}$ is distributed among n peers $S_1, S_2, \ldots, S_n$. Any entry of **X**, say $\sum_{j=1}^{|\mathcal{G}|}(x_{j1})^2$, can

be decomposed as

$$\sum_{j=1}^{|\mathcal{G}|}(x_{j1})^2 = \underbrace{\sum_{x_{j1}\in S_1}(x_{j1})^2}_{\text{for } S_1} + \underbrace{\sum_{x_{j1}\in S_2}(x_{j1})^2}_{\text{for } S_2} + \ldots + \underbrace{\sum_{x_{j1}\in S_n}(x_{j1})^2}_{\text{for } S_n}$$

Therefore each entry of $\mathbf{X}$ and $\mathbf{Y}$ can be computed by simple sum over all the peers. Thus, instead of sending the raw data in the convergecast round, peer P_i can forward a locally computed matrix $\mathbf{X}_i$ and $\mathbf{Y}_i$. Peer P_j, on receiving this, can forward a new matrix $\mathbf{X}_j$ and $\mathbf{Y}_j$ by aggregating, in a component-wise fashion, its local matrix and the received ones. Note that the avoidance of the sampling technique ensures that the result is exactly the same compared to a centralized setting.

Communication Complexity Next we prove a lemma which states the communication complexity of computing the linear regression model.

Theorem 3.8.1. *The communication complexity of computing a linear regression model is only dependent on the degree of the polynomial (d) and is independent of the number of data points i.e. $|\mathcal{G}|$.*

Proof. As shown in Section 3.8.4, the task of computing the regression coefficients $\{a_0, a_1, \ldots a_{d-1}\}$ can be reduced to computing the matrices $\mathbf{X}_i$ and $\mathbf{Y}_i$. The dimensionality of $\mathbf{X}_i$ $d.d = d^2$. Similarly the dimensionality of $\mathbf{Y}_i$ $d.1 = d$. Therefore the total communication complexity is $O(d^2)$, which is independent of the size of the dataset $|\mathcal{G}|$. □

The efficiency of the convergecast process is due to the fact that $d \ll |\mathcal{G}|$. Hence there can be significant savings in terms of communication by not communicating the raw data.

3.9 Experimental Validation

To validate the performance of our algorithms we conducted experiments on a simulated network of thousands of peers. In this section we discuss the experimental setup and analyze the performance of the algorithms.

3.9.1 Experimental Setup

Our implementation makes use of the Distributed Data Mining Toolkit (DDMT) [44] — a distributed data mining development environment from DIADIC research lab at UMBC. DDMT uses topological information which can be generate by BRITE [23], a universal topology generator from Boston University. In our simulations we used topologies generated according to the *Barabasi Albert (BA)* model, which is often considered a reasonable model for the Internet. BA also defines delays for network edges, which are the basis for our time measurement[1]. On top of the network generated by BRITE, we overlayed a communication tree.

Data Generation The data used in the simulations of the L2 Thresholding algorithm, the Mean Monitoring algorithm, the k-Means and the Eigen Monitoring algorithm was generated using a mixture of Gaussians in $\mathbb{R}^d$. Every time a simulated peer needed an additional data point, it sampled d Gaussians and multiplied the resulting vector with a $d \times d$ covariance matrix in which the diagonal elements were all 1.0's while the off-diagonal elements were chosen uniformly between 1.0 and 2.0. Alternatively, 10% of the points were chosen uniformly at random in the range of $\mu \pm 3\sigma$. At controlled intervals, the means of the Gaussians were changed, thereby creating an epoch change. A typical data in two dimensions can be seen in Figure 3.9. We preferred synthetic data because of the large number of factors (twelve, in our analysis) which influence the behavior of an

[1] Wall time is meaningless when simulating thousands of computers on a single PC.

algorithm, and the desire to perform a tightly controlled experiment in order to understand the behavior of a complex algorithm which operates in an equally as complex environment.

For the multivariate regression problem, the input data of a peer is a vector $(x_1, x_2, \ldots, x_d) \in \mathbb{R}^d$, where the first $d-1$ dimensions correspond to the input variables and the last dimension corresponds to the output. We have conducted experiments on both linear and non-linear regression models. For the linear model, the output is generated according to $x_d = a_0 + a_1x_1 + a_2x_2 + \ldots + a_{d-1}x_{d-1}$. We have used two functions for the non-linear model: (1) $x_3 = a_0 + a_1a_2x_1 + a_0a_1x_2$ (multiplicative) and (2) $x_3 = a_0 * sin(a_1 + a_2x_1) + a_1 * sin(a_2 + a_0x_2)$ (sinusoidal). Every time a simulated peer needs an additional data point, it chooses the values of $x_1, x_2, \ldots x_{d-1}$, each independently in the range -100 to +100. Then it generates the value of the target variable x_d using any of the above functions and adds a uniform noise in the range 5 to 20% of the value of the target output. The regression weights $a_0, a_1, \ldots, a_{d-1}$'s are changed randomly at controlled intervals to create an epoch change.

Measurement Metric The two most important qualities measured in our experiments are the *quality* of the result and the *cost* of the algorithm. Quality is defined differently for the L2 Thresholding algorithm, the Mean Monitoring algorithm, the k-Means, the Eigen Monitoring algorithm and the Multivariate Regression algorithm.

For the L2 Thresholding algorithm, quality is measured in terms of the number of peers correctly computing an alert *i.e.* the percentage of peers for whom $\left\|\overrightarrow{\mathcal{K}_i}\right\| < \epsilon$ when $\left\|\overrightarrow{\mathcal{G}}\right\| < \epsilon$, and the percentage of peers for whom $\left\|\overrightarrow{\mathcal{K}_i}\right\| \geq \epsilon$ when $\left\|\overrightarrow{\mathcal{G}}\right\| \geq \epsilon$. We measure the maximal, average and minimal quality over all the peers (averaged over a number of different experiments). Quality is reported in three different scenarios: overall quality, averaged over the entire experiment; and quality on stationary data, measured separately for periods in which the mean of the data is inside the ϵ-circle $\left(\left\|\overrightarrow{\mathcal{G}}\right\| < \epsilon\right)$ and for periods in which the means of the data is outside the circle $\left(\left\|\overrightarrow{\mathcal{G}}\right\| \geq \epsilon\right)$.

For the Mean Monitoring algorithm, quality is the average distance between $\overrightarrow{\mathcal{G}}$ and the computed mean vector $\overrightarrow{\mu}$. We plot, separately, the overall quality (during the entire experiment) and the quality after the sufficient statistics have been collected.

For the k-Means and eigenvector monitoring algorithm, quality is defined as the distance between the solution of our algorithm and that computed by a centralized algorithm, given all the data of all of the peers.

Lastly, for the regression monitoring algorithm, quality is defined as the L2 norm distance between the solution of our algorithm and the actual regression weights.

We measured the cost of the algorithm according to the frequency in which messages are sent by each peer. Because of the leaky bucket mechanism which is part of the algorithm (see Section 3.7.1 for explanation), the rate of messages per average peer is bounded by two for every L time units (one to each neighbor, for an average of two neighbors per peer). The trivial algorithm that floods every change in the data would send messages at this rate. The communication cost of our algorithms is thus defined in terms of normalized messages - the portion of this maximal rate which the algorithm uses. Thus, 0.1 normalized messages means that nine times out of ten the algorithm manages to avoid sending a message. We report both overall cost, which includes the stationary and transitional phases of the experiment (and thus is necessarily higher), and the monitoring cost, which only refers to stationary periods. The monitoring cost is the cost paid by the algorithm even if the data remains stationary; hence, it measures the "wasted effort" of the algorithm. We also separate, where appropriate, messages pertaining to the computation of the L2 Thresholding algorithm from those used for convergecast and broadcast of statistics.

Typical Experiment There are many factors which may influence the performance of the algorithms. First, are those pertaining to the data: the number of dimensions d, the covariance σ, and the distance between the means of the Gaussians of the different epochs (the algorithm is oblivious to the actual values of the means), and the length of the epochs

T. Second, there are factors pertaining to the system: the topology, the number of peers, and the size of the local data. Last, there are control arguments of the algorithm: most importantly ϵ — the desired alert threshold, and then also L — the maximal frequency of messages. In all the experiments that we report in this section, one parameter of the system was changed and the others were kept at their default values. The default values were : number of peers = 1000, $|S_i| = 800$, $\epsilon = 2$, $d = 5$, $L = 500$ (where the average edge delay is about 1100 time units), and the Frobenius norm of the covariance of the data $\|\sigma\|_F = \sum_{i=1...m} \sum_{j=1...n} |\sigma_{i,j}|^2$ at 5.0. We selected the distance between the means so that the rates of false negatives and false positives are about equal. More specifically, the means for one of the epochs was +2 along each dimension and for the other it was -2 along each dimension. For each selection of the parameters, we ran the experiment for a long period of simulated time, allowing 10 epochs to occur.

A typical experiment is described in Figure 3.10(a) and 3.10(b). In the experiment, after every 2×10^5 simulator ticks, the data distribution is changed, thereby creating an epoch. To start with, every peer is given the same mean as the mean of the Gaussian. Thus a very high percentage ($\sim$ 100 %) of the peers states that $\left\| \overrightarrow{\mathcal{G}} \right\| < \epsilon$. After the aforesaid number (2×10^5) of simulator ticks, we change the Gaussian without changing the mean given to each peer. Thus, for the next epoch, we see that a very low percentage of the peers ($\sim$ 0 %) output that $\left\| \overrightarrow{\mathcal{G}} \right\| < \epsilon$. For the cost of the algorithm in Figure 3.10(b), we see that messages exchanged during the stationary phase are low. Many messages are, however, exchanged as soon as the epoch changes. This is expected since all the peers need to communicate in order to get convinced that the distribution has indeed changed. The number of messages decreases once the distribution becomes stable again.

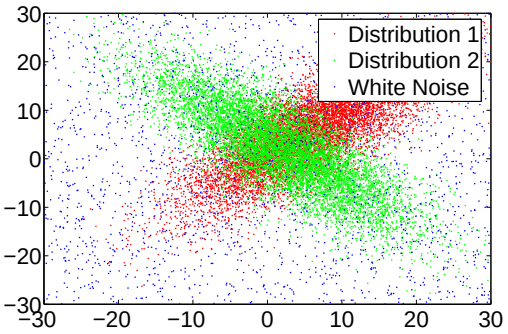
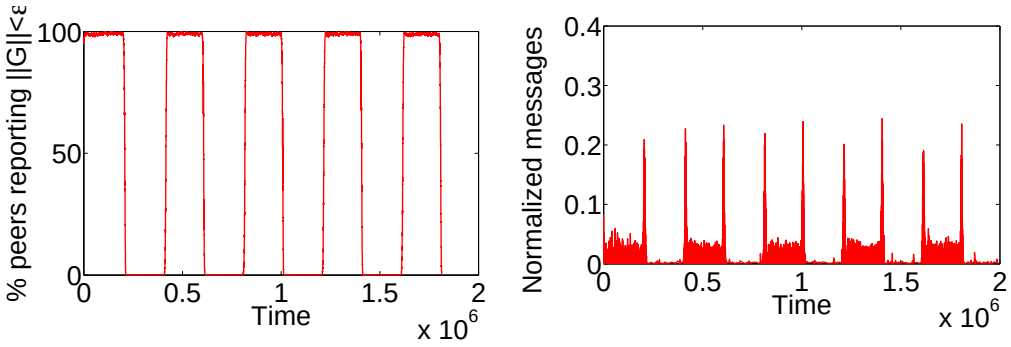


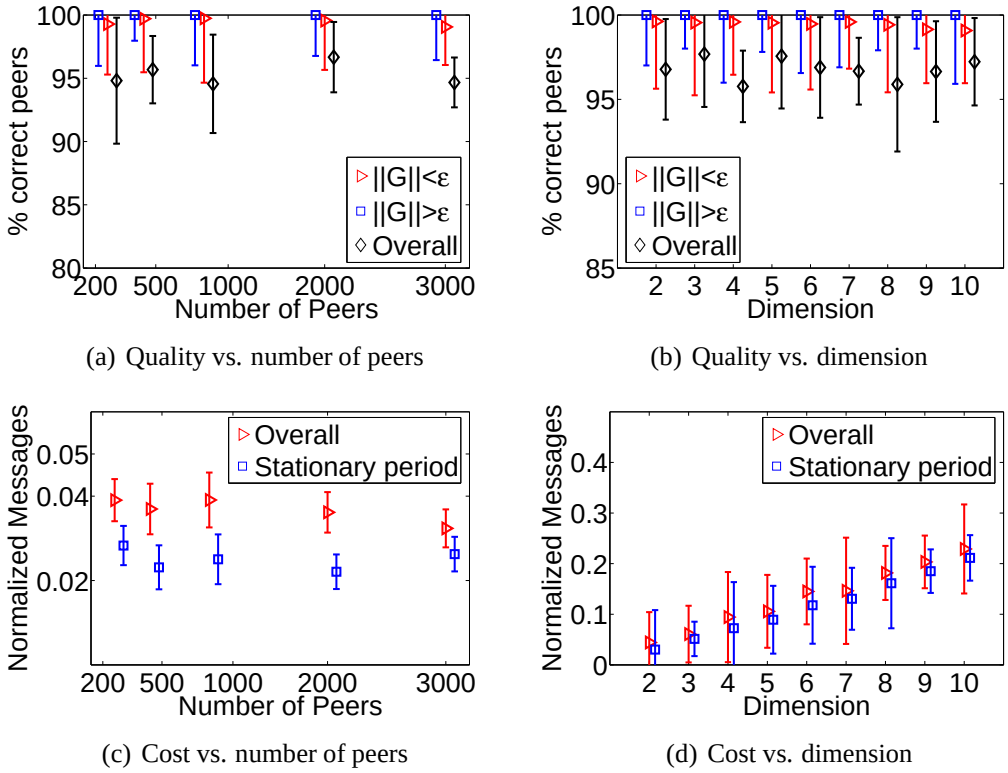
FIG. 3.9. 2-D typical data used in the experiment. It shows the two gaussians along with random noise.



(a) Typical changes in the percent of peers with $\left\|\vec{\mathcal{K}}_i\right\| \geq \epsilon$

(b) Typical messaging throughout an experiment

FIG. 3.10. A typical experiment is run for 10 equal length epochs. The epochs have very similar means. Quality and overall cost are measured across the entire experiment — including transitional phases. The monitoring cost is measured on the last 80% of every epoch, in order to ignore transitional effects.



(a) Quality vs. number of peers

(b) Quality vs. dimension

(c) Cost vs. number of peers

(d) Cost vs. dimension

FIG. 3.11. Scalability of Local L2 algorithm w.r.t number of peers and the dimension. Both quality and cost w.r.t number of peers converge to a constant, showing high scalability. For dimension variability, the cost increases almost linearly w.r.t dimension, when the space increases exponentially.

3.9.2 Experiments with Local L2 Thresholding Algorithm

The L2 Thresholding algorithm is the simplest one we present here. In our experiments, we use the L2 Thresholding to establish the scalability of the algorithms with respect to both the number of peers and the dimensionality of the data, and the dependency of the algorithm on the main parameters — the norm of the covariance σ, the size of the local data set, the tolerance ϵ, and the bucket size L.

In Figure 3.11, we analyze the scalability of the local L2 algorithm. As Figure 3.11(a) and Figure 3.11(c) show, the average quality and cost of the algorithm converge to a constant as the number of peers increase. This typifies local algorithms — because the computation is local, the total number of peers do not affect performance. Hence, there could be no deterioration in quality or cost. Also the number of messages per peer become a constant — typical to local algorithms. Similarly, Figure 3.11(b) and Figure 3.11(d) show the scalability with respect to the dimension of the problem. As shown in the figures, quality does not deteriorate when the dimension of the problem is increased. Also note that the cost increases approximately linearly with the dimension. This independence of the quality can be explained if one thinks of what the algorithm does in terms of domain linearization. We hypothesis that when the mean of the data is outside the circle, most peers tend to select the same half-space. If this is true then the problem is projected along the vector defining that half-space — i.e., becomes uni-dimensional. Inside the circle, the problem is again uni-dimensional; if thought about in terms of the polar coordinate system (rooted at the center of the circle), then obviously the only dimension on which the algorithm depends is the radius. The dependency of the cost on the dimension stems from the linear dependence of the variance of the data on the number of Gaussians with whose variance is constant.

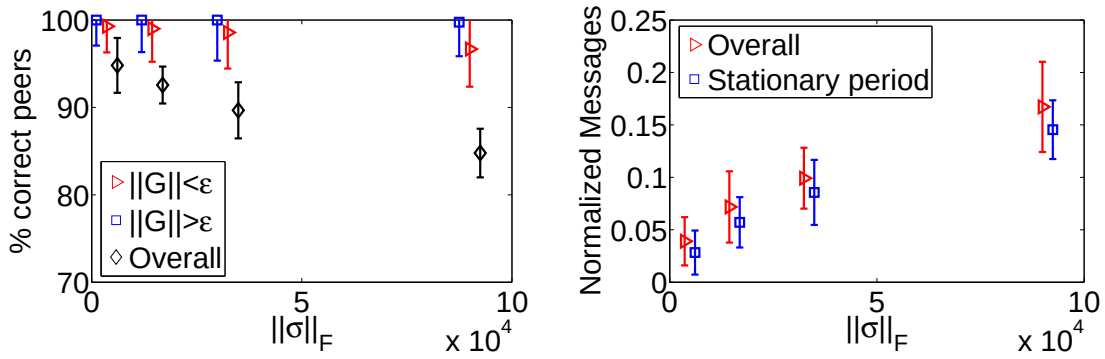
In Figures 3.12—3.15, we explore the dependency of the L2 algorithm on different parameters *viz.* Frobenius norm of the covariance of the data σ ($\|\sigma\|_F = \sum_{i=1\ldots m}\sum_{j=1\ldots n}|\sigma_{i,j}|^2$), the size of the local dataset $|S_i|$, the alert threshold ϵ, and the

size of the leaky bucket L. As noted earlier, in each experiment one parameter was varied and the rest were kept at their default values.

The first pair of figures, Figure 3.12(a) and Figure 3.12(b), outline the dependency of the quality and the cost on the covariance of the data ($\sigma = A\overrightarrow{E}$) where A is the covariance matrix and $\overrightarrow{E}$ is the variance of the Gaussians (refer to Section 3.9.1). Matrix A is as defined in Section 3.9.1 while $\overrightarrow{E}$ is the column vector representing the variance of the Gaussians and takes the values 5, 10, 15 or 25. For epochs with $\left\|\overrightarrow{\mathcal{G}}\right\| < \epsilon$, the maximal, the average, and the minimal quality in every experiment decrease linearly with the variance (from around 99% on average to around 96%). Epochs with $\left\|\overrightarrow{\mathcal{G}}\right\| > \epsilon$, on the other hand, retained very high quality, regardless of the level of variance. The overall quality also decreases linearly from around 97% to 84%, apparently resulting from slower convergence on every epoch change. As for the cost of the algorithm, this increases as the square root of $\|\sigma\|_F$ (i.e., linear to the variance), both for the stationary and overall period. Nevertheless, even with the highest variance, the cost stayed far from the theoretical maximum of two messages per peer per leaky bucket period.

The second pair of figures, Figure 3.13(a) and Figure 3.13(b), show that the variance can be controlled by increasing the local data. As $|S_i|$ increases, the quality increases, and cost decreases, proportional to $\sqrt{|S_i|}$. The cause of that is clearly the relation of the variance of an i.i.d. sample to the sample size which is inverse of the square root.

The third pair of figures, Figure 3.14(a) and Figure 3.14(b), present the effect of changing ϵ on both the cost and quality of the algorithm. As can be seen, below a certain point, the number of false positives grows drastically. The number of false negatives, on the other hand, remains constant regardless of ϵ. When ϵ is about two, the distances of the two means of the data (for the two epochs) from the boundary of the circle are approximately the same and hence the rates of false positives and false negatives are approximately the same too. As ϵ decreases, it becomes increasingly difficult to judge if the mean of the data is inside



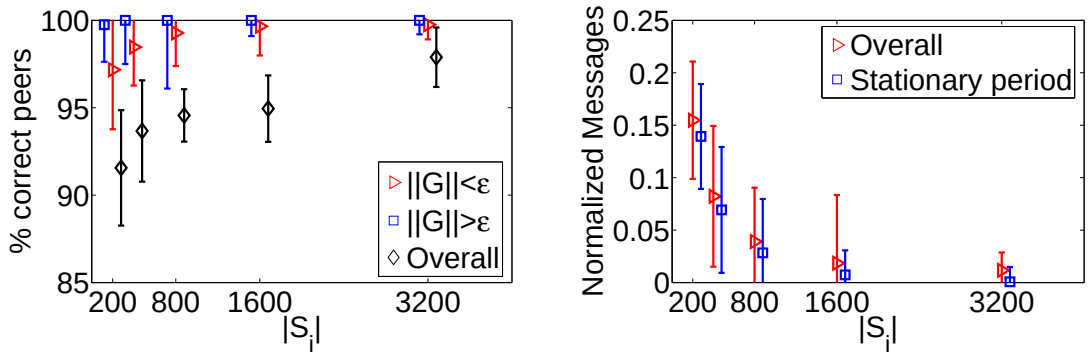
(a) Quality vs. $\|\sigma\|_F$ (b) Cost vs. $\|\sigma\|_F$

FIG. 3.12. Dependency of cost and quality of L2 Thresholding on $\|\sigma\|_F$. Quality is defined as the percentage of peers correctly computing an alert (separated for epochs with $\left\|\vec{\mathcal{G}}\right\|$ less and more than ϵ). Cost is defined as the portion of the leaky buckets intervals that are used. Both overall cost and cost of just the stationary periods are reported. Quality degrades and cost increases as $\|\sigma\|_F$ is increased.

the smaller circle and increasingly easier to judge that the mean is outside the circle. Thus, the number of false positives increase. The cost of the algorithm decreases linearly as ϵ grows from 0.5 to 2.0, and reaches nearly zero for $\epsilon = 3$. Note that even for a fairly low $\epsilon = 0.5$, the number of messages per peer per leaky bucket period is around 0.75, which is far less than the theoretical maximum of 2.

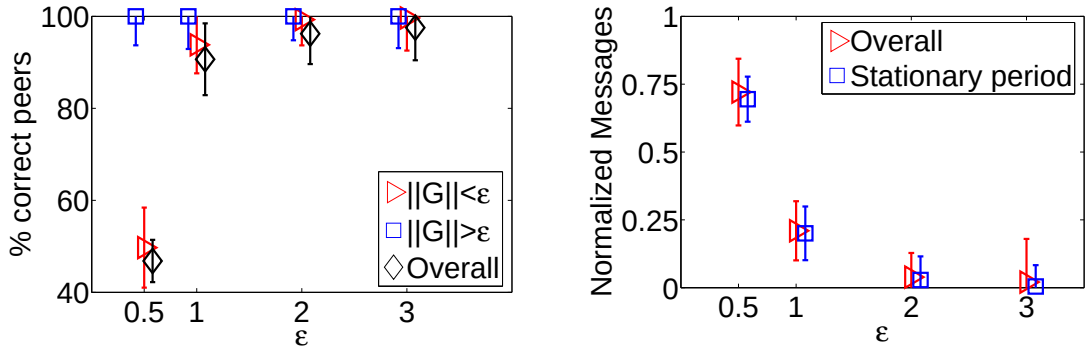
Figure 3.15(a) and Figure 3.15(b) explore the dependency of the quality and the cost on the size of the leaky bucket L. Interestingly, the reduction in cost here is far faster than the reduction in quality, with the optimal point (assuming 1:1 relation between cost and quality) somewhere between 100 time units and 500 time units. It should be noted that the average delay BRITE assigned to and edge is around 1100 time units. This shows that even a very permissive leaky bucket mechanism is sufficient to greatly limit the number of messages.

We conclude that the L2 Thresholding provides a moderate rate of false positives even



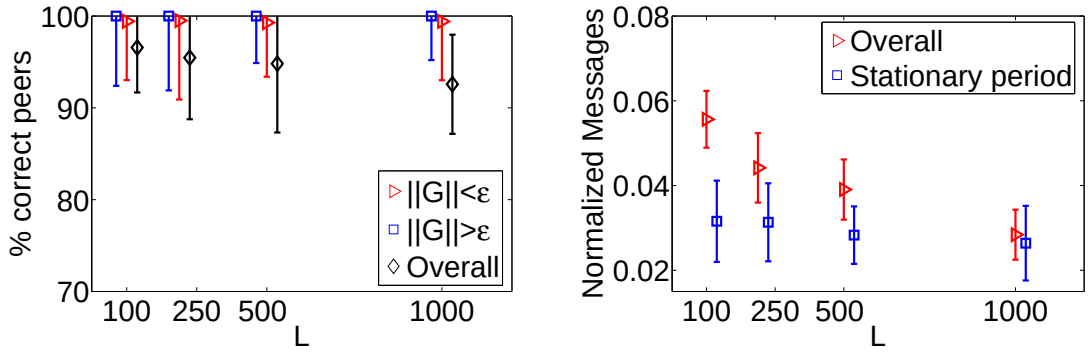
(a) Quality vs. $|S_i|$ (b) Cost vs. $|S_i|$

FIG. 3.13. Dependency of cost and quality of L2 Thresholding on $|S_i|$. Quality is defined as the percentage of peers correctly computing an alert (separated for epochs with $\left\|\vec{\mathcal{G}}\right\|$ less and more than ϵ). Cost is defined as the portion of the leaky buckets intervals that are used. Both overall cost and cost of just the stationary periods are reported. The quality improves and the cost decreases as $|S_i|$ is increased.



(a) Quality vs. ϵ (b) Cost vs. ϵ

FIG. 3.14. Dependency of cost and quality of L2 Thresholding on ϵ. Quality is defined as the percentage of peers correctly computing an alert (separated for epochs with $\left\|\vec{\mathcal{G}}\right\|$ less and more than ϵ). Cost is defined as the portion of the leaky buckets intervals that are used. Both overall cost and cost of just the stationary periods are reported. There is a drastic improvement of quality and decrease in cost as ϵ is increased.



(a) Quality vs. L (b) Cost vs. L

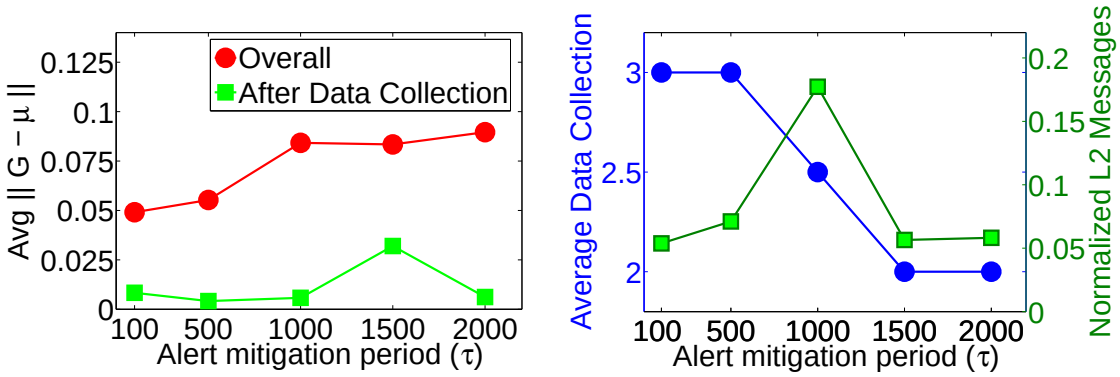
FIG. 3.15. Dependency of cost and quality of L2 Thresholding the size of the leaky bucket L. Quality is defined as the percentage of peers correctly computing an alert (separated for epochs with $\left\|\vec{\mathcal{G}}\right\|$ less and more than ϵ). Cost is defined as the portion of the leaky buckets intervals that are used. Both overall cost and cost of just the stationary periods are reported. Quality is almost unaffected by L, while cost decreased with increasing L.

for noisy data and an excellent rate of false negatives regardless of the noise. It requires little communication overhead during stationary periods. Furthermore, the algorithm is highly scalable — both with respect to the number of peers and dimensionality — because performance is independent of the number of peers and dimension of the problem.

3.9.3 Experiments with Means-Monitoring

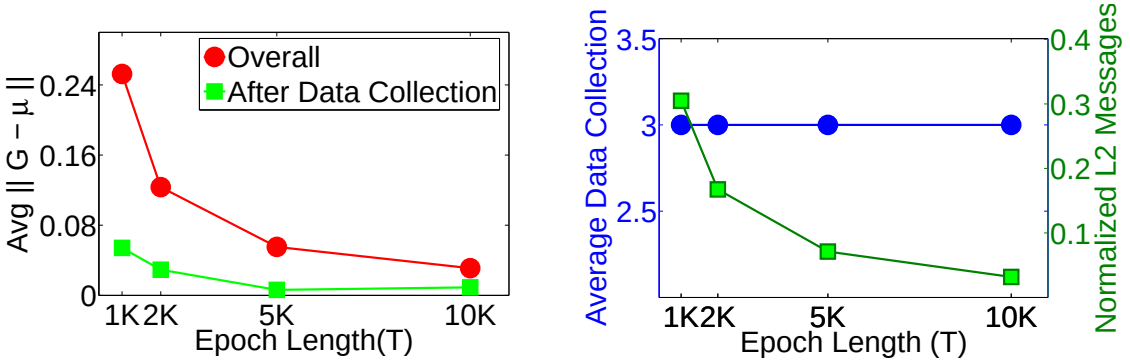
Having explored the effects of the different parameters of the L2 Thresholding algorithm, we now shift our focus on the experiments with the Mean Monitoring algorithm. We have explored the three most important parameters that affect the behavior of the Mean Monitoring algorithm: τ — the alert mitigation period, T — the length of an epoch, and ϵ — the alert threshold.

Figures 3.16—3.18 summarize the results of these experiments. As can be seen, the quality, measured by the distance of the actual means vector $\overrightarrow{\mathcal{G}}$ from the computed one $\overrightarrow{\mu}$



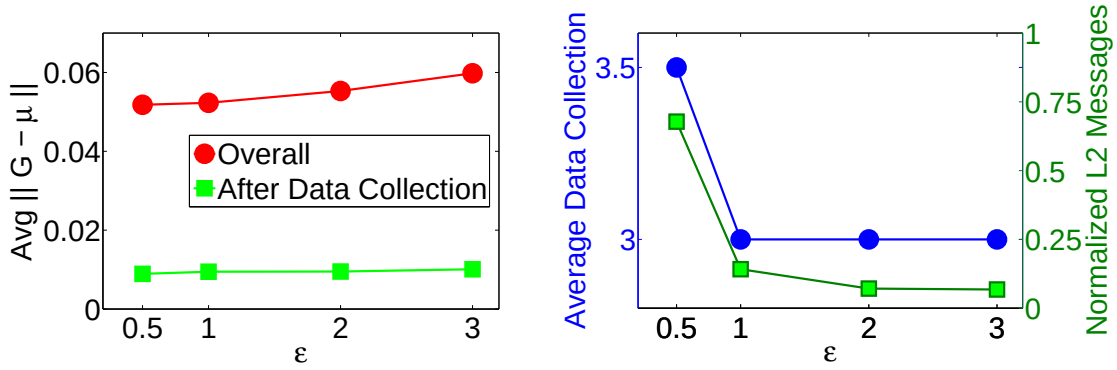
(a) Quality vs. τ (b) Cost vs. τ

FIG. 3.16. Dependency of cost and quality of Mean-Monitoring on alert mitigation period τ. The number of data collection rounds decrease as τ is increased.



(a) Quality vs. Epoch Length (b) Cost vs. Epoch Length

FIG. 3.17. Dependency of cost and quality of Mean-Monitoring on epoch length (T). As T increases, the average $||\mathcal{G} - \mu||$ and L2 monitoring cost decrease since the average is taken over a larger stationary period in which μ is correct.



(a) Quality vs. ϵ (b) Cost vs. ϵ

FIG. 3.18. Dependence of cost and quality of Mean-Monitoring on ϵ. Average $||\mathcal{G} - \mu||$ increases and the cost decreases as ϵ is increased.

is excellent in all three graphs. Also shown are the cost graphs with separate plots for the L2 messages (on the right axis) and the number of convergecast rounds — each costing two messages per peer on average — (on the left axis) per epoch. In Figure 3.16(a), the average distance between $\overrightarrow{\mathcal{G}}$ and $\overrightarrow{\mu}$ decreases as the alert mitigation period (τ) is decreased for the entire length of the experiment. This is as expected, since, with a smaller τ, the peers can rebuild the model more frequently, resulting in more accurate models. On the other hand, the quality after the data collection is extremely good and is independent of τ. With increasing τ, the number of convergecast rounds per epoch decreases (from three to two on average) as shown in Figure 3.16(b). In our analysis, this results from a decrease in the number of false alerts.

Figure 3.17(a) depicts the relation of the quality (both overall and stationary periods) to T. The average distance between the estimated mean vector and the actual one decreases as the epoch length T increases. The reason is the following: at each epoch, several convergecast rounds usually occur. The later the round is, the less polluted is the data by remnants of the previous epoch — and thus the more accurate is $\overrightarrow{\mu}$. Thus, when the epoch length increases, the proportion of these later $\overrightarrow{\mu}$'s, which are highly accurate, increases in

the overall quality leading to a more accurate average. Figure 3.17(b) shows a similar trend for the cost incurred. One can see that the number of L2 messages decrease as T increases. Clearly, the more accurate $\overrightarrow{\mu}$ is, the less monitoring messages are sent. Therefore with increasing T, the quality increases and cost decreases in the later rounds and these effects are reflected in the figures.

Finally, the average distance between $\overrightarrow{\mathcal{G}}$ and $\overrightarrow{\mu}$ decreases as ϵ decreases as demonstrated in Figure 3.18(a). This is as expected, since with decreasing ϵ, the L2 algorithm ensures that these two quantities be brought closer to each other and thus the average distance between them decreases. The cost of the algorithm, however, shows the reverse trend. This result is intuitive — with increasing ϵ, the algorithm has bigger area in which to bound the global average and thus the problem becomes easier, and hence less costly, to solve.

On the whole, quality of the Mean-Monitoring algorithm outcome behaves well respective to all the three parameters influencing it. The monitoring cost *i.e.* L2 messages is also low. Furthermore, on an average, the number of convergecast rounds per epoch is around three — which can easily be reduced further by using a bigger τ as the default value.

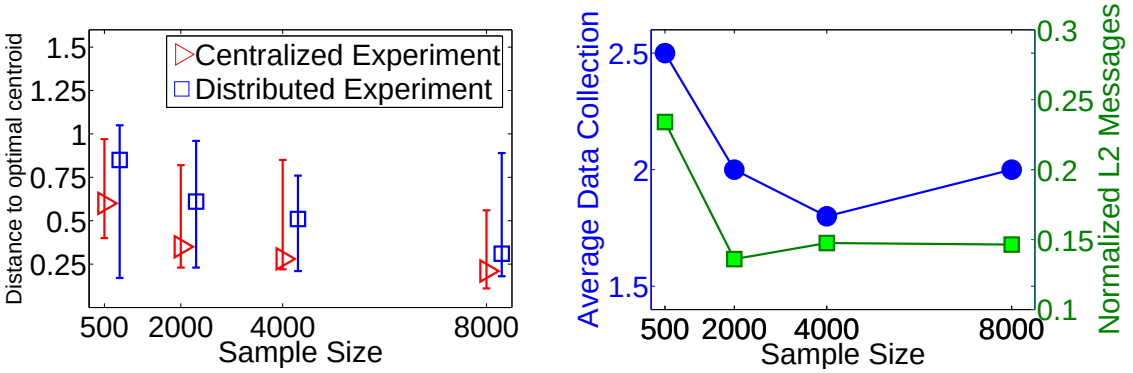
3.9.4 Experiments with k-Means and Eigen-Monitoring

In this set of experiments our goal is to investigate the effect of the sample size on both the P2P k-means algorithm and the Eigen-monitoring algorithm. To do that we compare the results of our algorithm to those of a centralized algorithm that processed the entire data. We compute the distance between each centroid computed by the P2P algorithm and the closest centroid computed by the centralized one. Since our algorithm is not only distributed but also sample-based, we include for comparison the results of centralized algorithm (k-means or eigenvector computation, respectively) which takes a sample from the entire data as its input. The most outstanding result, seen in Figure 3.19(a), is that most

of the error of the distributed algorithm is due to sampling and not due to decentralization. The error, both average, best case, and worst case, is very similar to that of the centralized sample-based algorithm. This is significant in two ways. First, the decentralized algorithm is obviously an alternative to centralization; especially considering the far lower communication cost. Secondly, the error of the decentralized algorithm can be easily controlled by increasing the sample size.

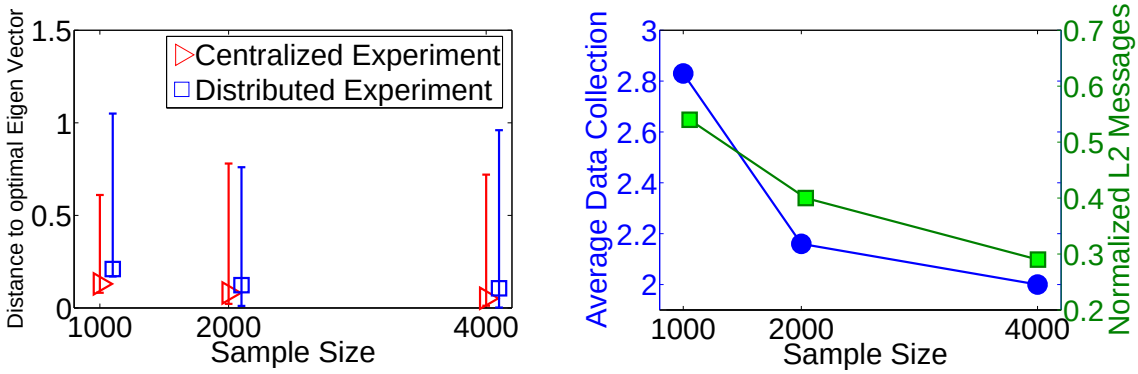
The costs of k-means have to be separated to those related to monitoring the current centroids and those related to the collection of the sample. Figure 3.19(b) presents the costs of monitoring a single centroid and the number of times data was collected per epoch. These could be multiplied by k to bound the total costs (note that messages relating to different centroids can be piggybacked on each other). The cost of monitoring decreases drastically with increasing sample size — resulting from the better accuracy provided by the larger sample. Also there is a decrease in the number of convergecast rounds as the sample size increases. The default value of the alert mitigation factor τ in this experimental setup was 500. For any sample size greater than 2000, the number of convergecast rounds is about two per epoch — in the first round, it seems, the data is so much polluted by data from the previous epoch that a new round is immediately triggered. As noted earlier, this can be further decreased using a larger value of τ.

Results for the Eigen-monitoring algorithm (Figure 3.20) are very similar to those of k-means. Even more evidently, the source of error in the eigenvector is that it is based on a sample from the data. Since the eigenvector is more robust statistics than k-means results, the error is significantly smaller. The monitoring cost and the data collection rounds also decrease with increasing sample size. Note that in this case we chose a smaller $\epsilon = 0.02$.



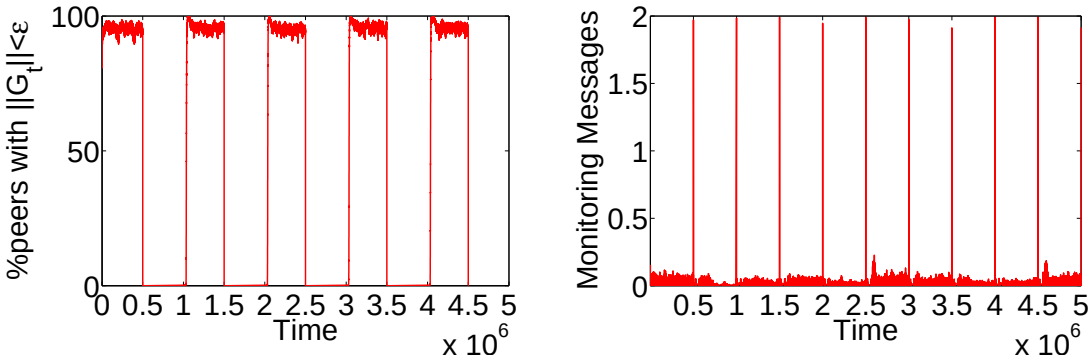
(a) Average Quality vs. Sample Size (b) Average Monitoring Cost vs. Sample Size

FIG. 3.19. The Quality and Cost of k-means Clustering.



(a) Average Quality vs. Sample Size (b) Average Monitoring Cost vs. Sample Size

FIG. 3.20. The Quality and Cost of Eigen monitoring.



(a) Percentage of peers with $\|\mathcal{K}_i\| < \epsilon$ (b) Monitoring Cost

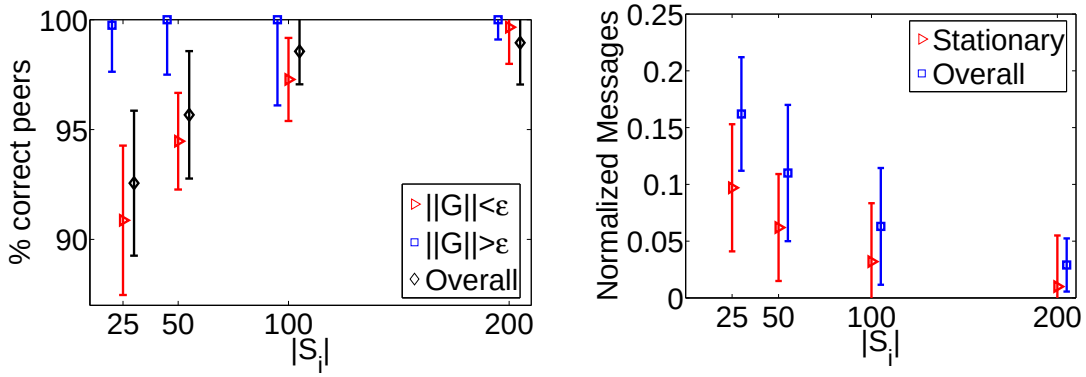
FIG. 3.21. A typical experiment for the regression monitoring algorithm.

3.9.5 Experiments with Multivariate Regression Algorithm

We divide the experimental results in two subsections — the first part consists of the results related to the regression monitoring algorithm and the second section presents the results related to the regression computation problem.

Results: Regression Monitoring A typical experiment for the regression monitoring algorithm is shown in Figure 3.21. As before, the monitoring cost is low and quality is good throughout the execution of the experiment.

There are four external parameters which can influence the behavior of the regression monitoring algorithm: size of local dataset $|S_i|$, the radius of the circle ϵ, size of the leaky bucket L and noise in the data. Apart from these there are also the system size (number of peers) and dimensionality of the multivariate regression problem which can affect performance. In this section we present the quality (inside *i.e.* $\left|\left|\overrightarrow{\mathcal{G}}\right|\right| < \epsilon$, outside *i.e.* $\left|\left|\overrightarrow{\mathcal{G}}\right|\right| > \epsilon$ and overall) and cost of the algorithm w.r.t. different parameters. Note that, unless otherwise stated, we have used the following default values for the different parameters: number of peers = 1000, $|S_i| = 50$, $\epsilon = 1.5$, $d = 10$ and $L = 500$ (where the average edge delay is

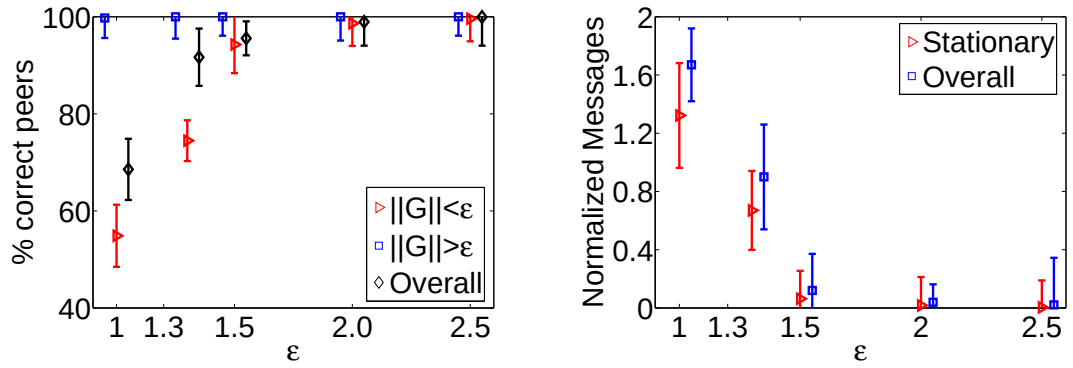


(a) Quality vs. $|S_i|$ (b) Cost vs. $|S_i|$

FIG. 3.22. Dependence of the quality and cost of the monitoring algorithm on $|S_i|$. As before, quality improves and cost decreases as $|S_i|$ increases.

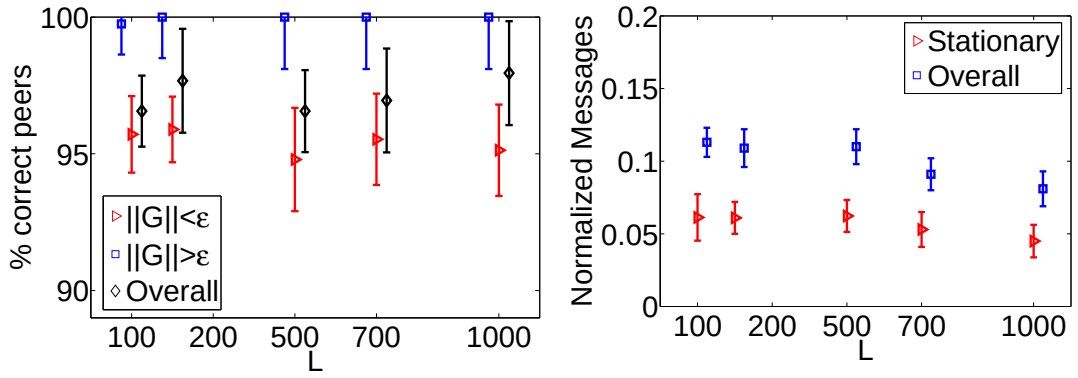
about 1100 time units). As we have already stated, independent of the regression function chosen, the underlying monitoring problem is always in $\mathbb{R}^2$. The results reported in this section are with respect to linear model since it is the most widely used regression model. Results of monitoring more complex models are reported in the next section.

Figures 3.22(a) and 3.22(b) show the quality and cost of the algorithm as the size of local dataset is changed. As expected, the inside quality increases and the cost decreases as the size of dataset increases. The outside quality is very high throughout. This stems from the fact that, with the noise in the data, it is easy for a peer to get flipped over when it is checking for inside a circle. On the other hand, noise cannot change the belief of the peer when the average is outside. In the second set of experiments, we varied ϵ from 1.0 to 2.5 (Figure 3.23(a) and 3.23(b)). Here also, the quality increases as ϵ is increased. This is because with increasing ϵ, there is a bigger region in which to bound the global average. This is also reflected with decreasing number of messages. Note that, even for $\epsilon = 1.0$, the normalized messages are around 1.6, which is far less than the theoretical maximum of 2 (assuming two neighbors per peer). The third set of experiments analyzes the effect of



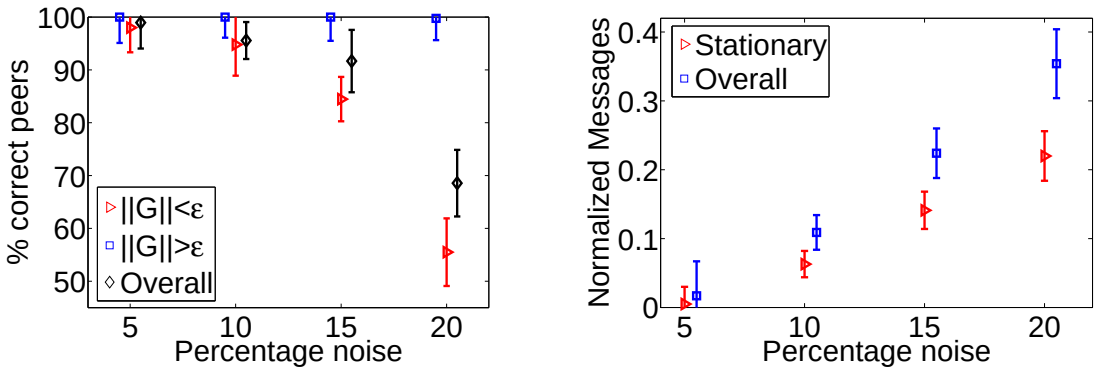
(a) Quality vs. ϵ (b) Cost vs. ϵ

FIG. 3.23. Dependence of the monitoring algorithm on ϵ. As ϵ is increased, quality improves and cost decreases since the problem becomes significantly simpler.



(a) Quality vs. L (b) Cost vs. L

FIG. 3.24. Behavior of the monitoring algorithm w.r.t size of the leaky bucket L. The quality and cost in almost invariant of L.



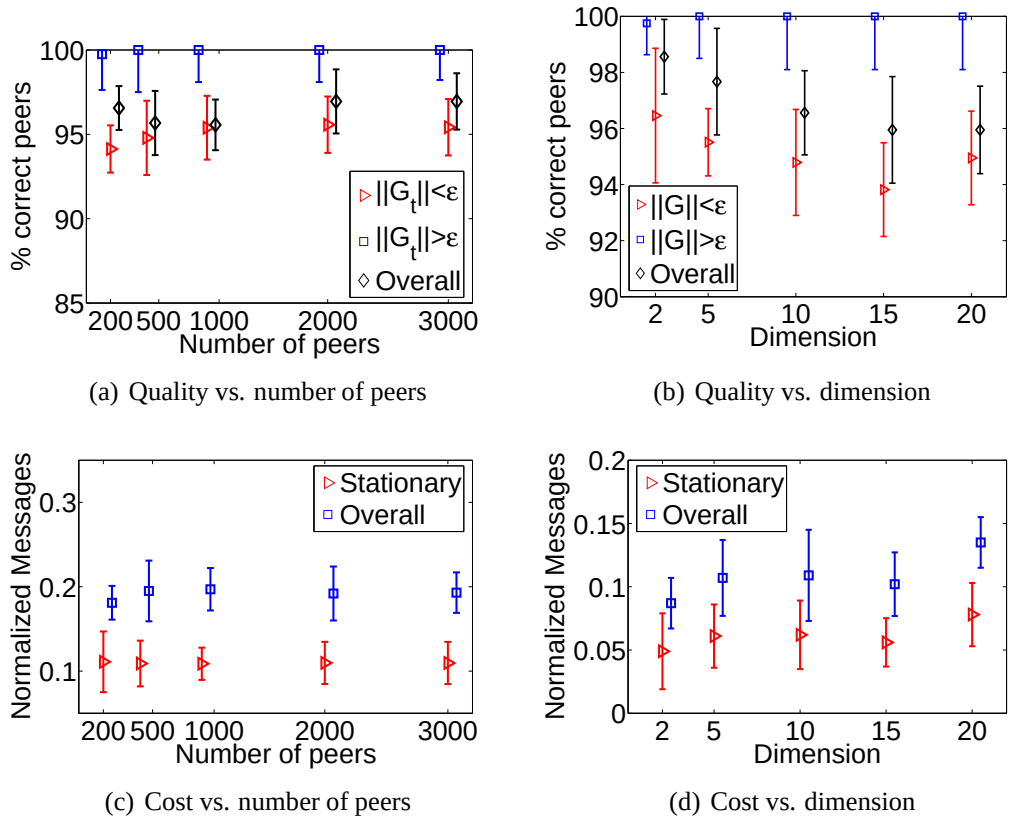
(a) Quality vs. Noise

(b) Cost vs. Noise

FIG. 3.25. Dependence of the quality and cost of the monitoring algorithm on noise in the data. The quality degrades and cost increases as percentage is increased.

leaky bucket L. As shown in Figure 3.24(a) quality does not depend on L, while Figure 3.24(b) shows that the cost decreases slowly with increasing L. Finally, Figures 3.25(a) and 3.25(b) depict the dependence of the noise on the monitoring algorithm. Quality degrades and cost increases with increasing noise. This is expected, since with increasing noise a peer is more prone to random effects. This effect can, however, be nullified by using a large dataset or bigger ϵ.

Our next experiment analyzes the scalability of the monitoring algorithm w.r.t the number of peers and dimension of the multivariate problem. As Figures 3.26(a) and 3.26(c) show, both the quality and cost of the algorithm converge to a constant as the number of peers increase. This is a typical behavior of local algorithms. For any peer, since the computation is dependent on the result from only a handful of its neighbors, the overall size of the network does not degrade the quality or cost. Similarly, Figures 3.26(b) and 3.26(d) show that the quality or the cost does not depend on the dimension of the multivariate problem either. This independence of the quality and cost can be explained by noting that the underlying monitoring problem is in $\mathbb{R}^2$. Therefore for a given problem, the system size



(a) Quality vs. number of peers

(b) Quality vs. dimension

(c) Cost vs. number of peers

(d) Cost vs. dimension

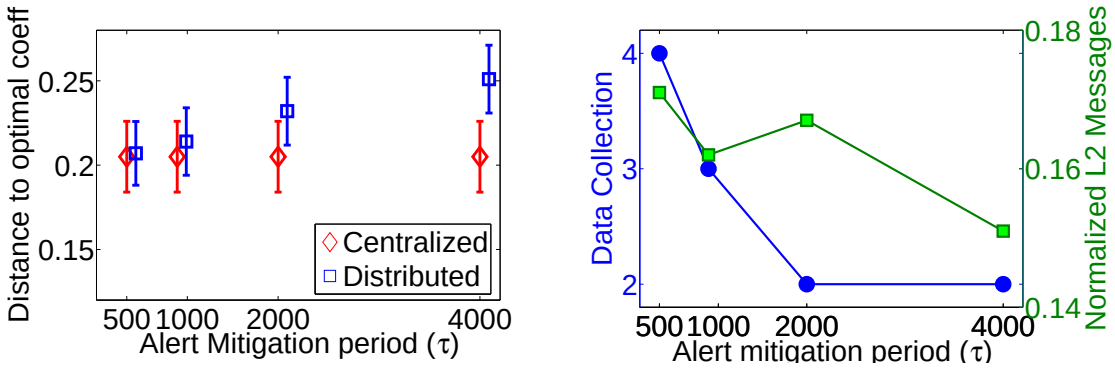
FIG. 3.26. Scalability with respect to both number of peers and dimension of the multivariate problem. Both quality and cost is independent of the number of peers and dimension of the multi-variate problem.

or dimensionality of the problem has no effect on the quality or the cost.

Overall, the results show that the monitoring algorithm offers extremely good quality, incurs low monitoring cost and has high scalability.

3.9.6 Results: Regression Models

Our next set of experiments measure the quality of the regression model computed by our algorithm against a centralized algorithm having access to the entire data. There are two important parameters to be considered here — (1) the alert mitigation constant (τ) and



(a) Quality for linear model

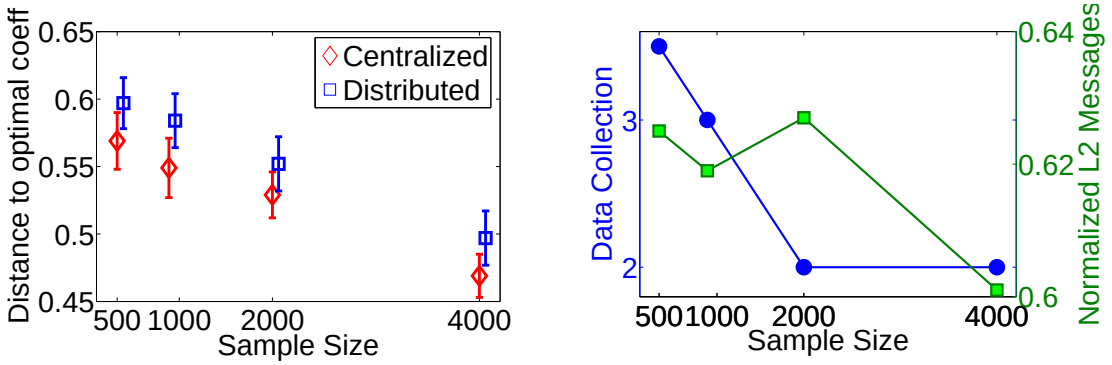
(b) Cost for linear model

FIG. 3.27. Quality and cost of computing regression coefficients for linear model.

(2) the sample size (for non-linear regression). For computing the non-linear regression coefficients, we have implemented the Nelder-Mead simplex method [138].

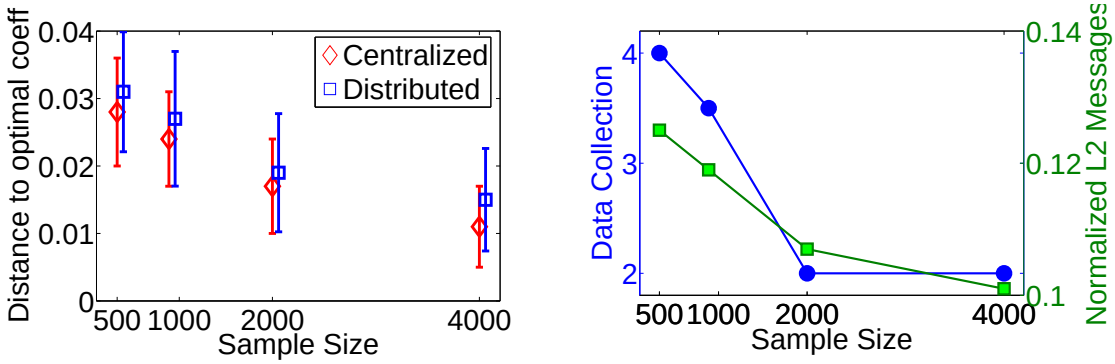
We have conducted experiments on three datasets. Each quality graph in Figures 3.27—3.29 presents two sets of error bars. The square markers show the L2 norm distance between the distributed coefficients and the actual ones. Also shown in each figure is the L2 norm distance between the coefficients found by a centralized algorithm and the actual ones (diamond markers). The first pair of figures, Figures 3.27(a) and 3.27(b) show the results of computing a linear regression model. Our aim is to measure the effect of variation of alert mitigation period τ on quality and cost. As shown in Figure 3.27(a), the quality of our algorithm deteriorates as τ increases. This is because, on increasing τ, a peer builds a model later and therefore is inaccurate for a longer intermediate period. Figure 3.27(b) shows that the number of data collection rounds (dot markers) decrease from four times to twice per epoch. This results from a decrease in the number of false alerts. Also shown are monitoring messages (green squares).

Figures 3.28(a) and 3.28(b) analyzes the quality of our algorithm while computing a non-linear multiplicative regression model *viz.* $x_3 = a_0 + a_1 a_2 x_1 + a_0 a_1 x_2$. Figure 3.28(a)



(a) Quality for multiplicative model (b) Cost for multiplicative model

FIG. 3.28. Quality and cost of computing regression coefficients for multiplicative model. The accuracy of the result becomes more accurate and the cost decreases as the sample size is increased.



(a) Quality for sinusoidal model (b) Cost for sinusoidal model

FIG. 3.29. Quality and cost of computing regression coefficients for sinusoidal model. The accuracy of the result becomes more accurate and the cost decreases as the sample size is increased.

presents the quality as other parameter *viz.* sampling size is varied. As expected, the results from the distributed and centralized computations converge with increasing sample size. Also the number of data collection rounds as depicted in Figure 3.28(b) decrease as sample size is increased.

The third pair of figures, Figures 3.29(a) and 3.29(b) show the same results for a sinusoidal model : $x_3 = a_0 * sin(a_1 + a_2 x_1) + a_1 * sin(a_2 + a_0 x_2)$. Here also the quality becomes better and the cost decreases as the sample size is increased.

To sum everything up, the regression computation algorithm offers excellent accuracy and low monitoring cost. Also, the number of convergecast-broadcast rounds is also two times per epoch on an average. We have tested our algorithm on several regression functions and the results are highly satisfactory.

3.10 Summary

In this chapter we have presented *DeFraLC* — the framework for deterministic data mining in P2P environments. The generic algorithm, *PeGMA* is capable of computing complex functions of the average data in large distributed system. We present a number of interesting applications of *PeGMA*: a local L2 thresholding algorithm, a reactive means-monitoring algorithm, a k-means algorithm, an eigenvector-monitoring algorithm and a multivariate regression algorithm. Besides the direct contributions of this work with respect to the calculation of L2 norm, k-means and eigen monitoring in P2P networks, it also suggests an entirely new way by which local algorithms can be used in data mining. Previously, developing a local algorithm for the direct computation of a data mining model was tedious. However, a small number of those local algorithms are arguably sufficient to provide tools for deciding whether a data mining model correctly represents the data. Models can therefore be computed by any method (exact, approximate, or even at random) and then efficiently judged by the local algorithm. Furthermore, whatever the costs are for

computing the model, they can be amortized on periods where the data distribution does not change and only the efficient local algorithm operates.

Chapter 4

DISTRIBUTED DECISION TREE INDUCTION IN PEER-TO-PEER SYSTEMS

4.1 Introduction

Decision tree induction [154][20] is a powerful statistical and machine learning tool widely used for data classification, predictive modeling and more. Given a set of learning examples (attribute values and corresponding class labels) at a single location, there exist several well-known methods to build a decision tree such as ID3 [154] and C4.5 [155]. However, there can be several situations in which the data is distributed over a large, dynamic network containing no special server or client peers. A typical example is Peer-to-Peer (P2P) networks. Performing data mining tasks such as building decision trees is very challenging in a P2P network because of the large number of data sources, the asynchronous nature of the P2P networks, and dynamic nature of the data. A scheme which centralizes the network data is unscalable because any change must be reported to the central site, since it might very well alter the result.

To deal with this, in this chapter we propose a P2P decision tree induction algorithm in which every peer learns and maintains the correct decision tree compared to a centralized scenario. Our algorithm is highly scalable, completely decentralized and asynchronous and adapts to changes in the data and the network. The algorithm is efficient in the sense that it

guarantees that as long as the decision tree represents the data, the communication overhead is low when compared to a broadcast-based algorithm. As a result, the algorithm is highly scalable. When the data distribution changes, the decision tree is updated automatically. Our work is the first of its kind in the sense that it induces decision trees in large P2P systems in a communication-efficient manner without the need for global synchronization and the tree is the same that would have been induced given all the data to all the peers.

In the next section we present the related work in distributed classification algorithms.

4.2 Related Work: Distributed Classification Algorithms

Classification is one of the classic problems of the data mining and machine learning fields. Researchers have proposed several solutions to this problem — Bayesian models [86], ID3 and C4.5 decision trees [154][155], and SVMs [192] being just a tiny selection. In this section we present several algorithms which have been developed for distributed classification. The solutions proposed by these algorithms differ in three major aspects — (1) how the search domain is represented using an objective function, (2) which algorithm is chosen to optimize the objective function, and (3) how the work is distributed for efficient searching through the entire space. The latter item has two typical modes — in some algorithms the learning examples are only used during the search for a function (e.g., in decision trees and SVMs) while in other they are also used during the classification of new samples (notably, in Bayesian classifiers).

Meta classifiers are another interesting group of classification algorithms. In a meta classification algorithm such as bagging [21] or boosting [62], many classifiers (of any of the previous mentioned kinds) are first built on either samples or partitions of the training data. Then, those “weak” classifiers are combined using a second level algorithm which can be as simple as taking the majority of their outcomes for any new sample.

Some classification algorithms are better suited for a distributed set up. For instance,

Stolfo et el. [176] learn a weak classifier on every partition of the data, and then centralize the classifiers and a sample of the data. This can be a lot cheaper than transferring the entire raw data. Then the meta-classifier is deduced centrally from these data. Another suggestion, by Bar-Or *et al.* [9] was to execute ID3 in a hierarchical network by centralizing, for every node of the tree and at each level, only statistics regarding the most promising attributes. These statistics can, as the authors show, provide a proof that the selected attribute is indeed the one having the highest gain — or otherwise trigger the algorithm to request further statistics.

Caragea *et al.* [29] presented a decision tree induction algorithm for both distributed homogenous and heterogenous environments. Noting that the crux of any decision tree algorithm is the use of an effective splitting criteria, the authors propose a method by which this criteria can be evaluated in a distributed fashion. More specifically, the paper shows that by only centralizing the summary statistics from each site to one location, there can be huge savings in terms of communication when compared to brute force centralization. Moreover, the distributed decision tree induced is the same compared to a centralized scenario. Their system is available as part of the INDUS system.

A different approach was taken by Giannella *et al.* [66] and Olsen [141] for inducing decision tree in vertically partitioned data. They used Gini information gain as the impurity measure and showed that Gini between two attributes can be formulated as a dot product between two binary vectors. To reduce the communication cost, the authors evaluated the dot product after projecting the vectors in a random smaller subspace. Instead of sending either the raw data or the large binary vectors, the distributed sites communicate only these projected low-dimensional vectors. The paper shows that using only 20% of the communication cost necessary to centralize the data, they can build trees which are at least 80% accurate compared to the trees produced by centralization.

Distributed probabilistic classification on heterogenous data sites have also been dis-

cussed by Merugu and Ghosh [120]. Similarly, Park *et al*. have proposed a fourier spectrum-based approach for decision tree induction in vertically partitioned datasets [147].

When the scale of the system grows to millions of partitions — as in most modern P2P systems — the quality of such algorithms degrade. This is because for such large scale systems, no centralization of data, statistics, or models, is practical any longer. By the time such statistics can be gathered, it is reasonable that both the data and the system have changed to the point that the model needs to be calculated again. Thus, classification in P2P networks requires a different breed of algorithms — ones that are fully decentralized, asynchronous and can deal well with dynamically changing data.

The work most related to the one described in this chapter is the Distributed Plurality Algorithm (DPV) by Ping *et al.* [116]. In that work, a meta classification algorithm is described in which every peer computes a weak classifier on its own data. Then weak classifiers are merged into a meta classifier by computing — per new sample — the majority of the outcomes of the weak classifiers. The computation of weak classifiers requires no communication overhead at all, and the majority is computed using an efficient local algorithm.

Our work is different from DPV in several ways: firstly, we compute an ID3-like decision tree from the entire data (rather than many weak classifiers). Because the entire data is used, smaller sub-populations of the data stand a chance to gather statistical significance and contribute to the model — therefore, we argue tat our algorithm can be, in general, more accurate. Secondly, as proposed in DPV, every peer needs to be aware of each new sample and provide their classification of it. This mode of operation, which is somewhat reminiscent of Bayesian classification, requires broadcasting the new samples to all peers or limits the algorithm to specific cases in which all peers cooperate in classification of new samples (given to all) based on their private past experience. In contrast, in our work, all peers jointly study the same decision tree. Then, when a peer is given a new sample, it can

be classified with no communication overhead at all. When learning samples are few and new samples are in abundance, our algorithm can be far more efficient.

4.3 Background

4.3.1 Assumptions

Let S denote a collection of data tuples with class labels that are horizontally distributed over a large (undirected) network of machines (peers) wherein each peer communicates only with its immediate neighbors (one hop neighbors) in the network. The communication network can be thought of as a graph with vertices (peers) V. For any given peer $k \in V$, let Γ_k denote the immediate neighbors of k. Peer k will only communicate directly with peers in Γ_k.

Our goal is to develop a distributed algorithm under which each peer computes the decision tree over S (the same tree at each peer). [1] However, the network is dynamic in the sense that the network topology can change i.e. peers may enter or leave at any time or the data held by each peer can change hence S, the union of all peers data, can be thought of as time-varying. Our distributed algorithm is designed to seamlessly adapt to network and data changes in a communication-efficient manner.

We assume that communication among neighboring peers is reliable and ordered and that when a peer is disconnected or reconnected its neighbors, k, are informed, *i.e.* Γ_k is known to k and is updated automatically. These assumptions can easily be enforced using standard numbering and retransmission in which messages are numbered, ordered and retransmitted if an acknowledgement does not arrive in time, ordering, and heart-beat mechanisms. Moreover, these assumptions are not uncommon and have been made elsewhere in the distributed algorithms literature [63]. Khilar and Mahapatra [100] discuss the

[1] Throughout this chapter, peers refer to the machines in the P2P network and nodes refer to the entries of the tree (in which the attribute is split).

use of heartbeat mechanisms for failure diagnosis in mobile ad-hoc networks.

Furthermore, for simplicity, we assume that the network topology forms a tree. This allows us to use a relatively simple distributed algorithm as our basic building block: distributed majority voting (more details later). We could get around this assumption in one of two ways. (1) Liss *et al.* [14], have developed an extended version of the distributed majority voting algorithm which does not require the assumption that the network topology forms a tree. We could replace our use of simple majority voting as a basic building block with the extended version developed by Liss *et al.* (2) The underlying tree communication topology could be maintained independently of our algorithm using standard techniques such as [63] (for wired networks) or [110] (for wireless networks).

4.3.2 Distributed Majority Voting

Our algorithm utilizes, as a building block, a variation of the distributed algorithm for *majority voting* developed by Wolff and Schuster [196]. Each peer $k \in V$ contains a real number δ^k and the objective is to determine whether $\Delta = \sum_{k \in V} \delta^k \geq 0$.

The following algorithm meets this objective. For peers $k, \ell \in V$, let $\delta^{k\ell}$ denote the most recent message (a real number) peer k sent to ℓ. Similarly $\delta^{\ell k}$ denotes the last message received by k from ℓ. Peer k computes $\Delta^k = \delta^k + \sum_{\ell \in \Gamma_k} \delta^{\ell k}$, which can be thought of as $k's$ estimate of Δ based on all the information available. Peer k also computes $\Delta^{k\ell} = \delta^{k\ell} + \delta^{\ell k}$, for each neighbor $\ell \in \Gamma_k$ which denotes the information exchanged between k and ℓ. When an event at peer k occurs, k will decide, for each neighbor ℓ, whether a message needs be sent to ℓ. An event at k consists of one of the following three situations: (i) k is initialized (enters the network or otherwise begins computation of the algorithm); (ii) k experiences a data change δ^k or a change of its neighborhood, Γ_k; (iii) k receives a message from a neighbor ℓ.

The crux of the algorithm is in determining when k must send a message to a neighbor

ℓ in response to k detecting an event. More precisely, the question is when can a message be avoided, despite the fact that the local knowledge has changed. Upon detecting an event, peer k would send a message to neighbor ℓ when either of the following two situations occurs: (i) k is initialized; (ii) $\left(\Delta^{k\ell} \geq 0 \wedge \Delta^{k\ell} > \Delta^{k}\right) \vee \left(\Delta^{k\ell} < 0 \wedge \Delta^{k\ell} < \Delta^{k}\right)$ evaluates true. Observe that since all events are local, the algorithm requires no form of global synchronization.

When k detects an event and the conditions above indicate that a message must be sent to neighbor ℓ, k sets $\delta^{k\ell}$ to $\alpha\Delta^{k} - \delta^{\ell k}$ (thereby making $\Delta^{k\ell} = \alpha\Delta^{k}$) and sends it to ℓ, where α is a fixed parameter between 0 and 1. If α were set close to one, then small subsequent variations in Δ^{k} will trigger more messages from k increasing the communication overhead. On the other hand, if α were set close to zero, the convergence rate of the algorithm could be made unacceptably slow. In all our experiments, we set α to 0.5. This mechanism replicates the one used by Wolff *et al.* in [195]. Detailed discussion on the value of α is presented in Appendix A. The pseudo-code is presented in Algorithm 10.

To avoid a message explosion, we implement a *leaky bucket* mechanism such that the interval between messages sent by a peer does not become arbitrarily small. This mechanism was also used by Wolff *et al.* in [195]. Each peer logs the time when the last message was sent. When a peer decides that a message need to be sent (to any of its neighbors), it does the following. If L time units has passed since the time the last message was sent, it sends the new message right away. Otherwise, it buffers the message and sets a timer to L units after the registered time the last message was sent. Once the timer expires all the buffered messages are sent. For the remainder of the chapter, we leave the leaky bucket mechanism implicit in our distributed algorithm descriptions.

Input: δ^k, N^k, L, α
Output: if $\Delta^k \geq 0$ then 1 else 0
Local variables: $\forall \ell \in N^k : \delta^{\ell k}, \delta^{k\ell}$
Definitions: $\Delta^k = \delta^k + \sum_{\ell \in N^k} \delta^{\ell k}$, $\Delta^{k\ell} = \delta^{k\ell} + \delta^{\ell k}$
Initialization:
begin
    forall $\ell \in N^k$ **do**
        $\delta^{k\ell} = \delta^{\ell k} = 0$;
        SendMessage(ℓ);
    end
end
if MessageRecvd$\big(P^\ell, \delta\big)$ **then** $\delta^{\ell k} \leftarrow \delta$;
if PeerFailure$\big(\ell \in N^k\big)$ **then** $N^k \leftarrow N^k \setminus \{\ell\}$;
if AddNeighbor$\big(\ell \in N^k\big)$ **then** $N^k \leftarrow N^k \cup \{\ell\}$;
if N^k, δ^k *changes or MessageRecvd* **then** call **OnChange**();
Function ***OnChange()***
begin
    forall $\ell \in N^k$ **do**
        if $\big(\Delta^{k\ell} \geq 0 \wedge \Delta^{k\ell} > \Delta^k\big) \vee \big(\Delta^{k\ell} < 0 \wedge \Delta^{k\ell} < \Delta^k\big)$ **then**
            call **SendMessage**(ℓ);
        end
    end
end
Function ***SendMessage(ℓ)***
begin
    if $time\,() - last\_message \geq L$ **then**
        $\delta^{k\ell} \leftarrow (\alpha\Delta^k - \delta^{\ell k})$;
        $last\_message \leftarrow time\,()$;
        Send $\big\langle \delta^{k\ell} \big\rangle$ to ℓ;
    end
    else Wait $L - (time\,() - last\_message)$ time units and then call **OnChange**();
end

Algorithm 10: Local Majority Vote

4.4 P2P Decision Tree Induction Algorithm

This section presents a distributed and asynchronous algorithm P^2DT which induces a decision tree over a P2P network in which every peer has a set of learning examples. P^2DT, which is inspired by ID3 and C4.5, aims to select at every node the attribute which will maximize a gain function. Then, P^2DT aims to split the node, and the learning examples associated with it, into two new leaf nodes and this process continues recursively. A stopping rule directs P^2DT to stop this recursion. In this section a simple depth limitation is used. Other, more complex predicates are described in Section 4.4.3.

The main computational task of P^2DT is choosing the attribute having the highest gain among all attributes. Similar to other distributed data mining algorithms, P^2DT needs to coordinate this decision among the multiple peers. The main exceptions of P^2DT are that it stresses the efficiency of decision making and the lack of synchronization. These features make it exceptionally scalable and therefore suitable for networks spanning millions of peers.

P^2DT deviates from the standard decision tree induction algorithms in the choice of a simpler gain function — the misclassification error — rather than the more popular (and, arguably, better) information-gain and gini-index functions. Misclassification error offers less distinction between attributes: a split can have the same misclassification error in these two seemingly different cases — (1) the erroneous examples are divided equally between the two leaves it creates or (2) if one of these leaves is 100% accurate. Comparatively, both information-gain and gini-index would prefer the latter case to the former. Still, the misclassification error can yield accurate decision trees (see, e.g., [185]) and its relative simplicity makes it far easier to compute in a distributed set-up.

In the interest of clarity, we divide the discussion of the algorithm into two subsections: Section 4.4.1 below describes an algorithm for the selection of the attribute offering the lowest misclassification error from amongst a large set of possibilities. Next, Section

4.4.2 describes how a collection of such decisions can be efficiently used to induce a decision tree.

4.4.1 Splitting Attribute Choosing using the Misclassification Gain Function

Notations Let S be a set of learning examples — each a vector in $\{0,1\}^d \times \{0,1\}$ — where the first d entries of each example denote the attributes, $A^1, \ldots, A^d$, and the additional one denotes the class $Class$. The cross table of attribute A^i and the class is $X^i = \begin{matrix} x^i_{00} & x^i_{01} \\ x^i_{10} & x^i_{11} \end{matrix}$ where x^i_{01} is the number of examples in the set S for which $A^i = 0$ and $Class = 1$. We also define the indicator variables $s^i_0 = sign\left(x^i_{00} - x^i_{01}\right)$ and $s^i_1 = sign\left(x^i_{10} - x^i_{11}\right)$, with $sign\left(x\right) = \begin{cases} 1 & x \geq 0 \\ -1 & x < 0 \end{cases}$.

We have used the gain measure based on misclassification gain (as described in [185] page 158 as an alternate to gini and entropy). For a binary attribute A^i, the misclassification impurity measure for $A^i = 0$ is $MI_0 = 1 - max\left(x^i_{00}, x^i_{01}\right)/x^i_{0.}$ (where $x^i_{0.}$ is the number of tuples with $A^i = 0$). Similarly, for $A^i = 1$ the misclassification impurity is $MI_1 = 1 - max\left(x^i_{10}, x^i_{11}\right)/x^i_{1.}$. Therefore, the misclassification gain (MG) for A^i is

$$\begin{aligned} MG &= I(parent) - (x^i_{0.}/|S|) \times [MI_0] - (x^i_{1.}/|S|) \times [MI_1] \\ &= I(parent) - (x^i_{0.}/|S|) + max\left(x^i_{00}, x^i_{01}\right)/|S| - (x^i_{1.}/|S|) + max\left(x^i_{00}, x^i_{01}\right)/|S| \\ &= I(parent) - 1 + max\left(x^i_{00}, x^i_{01}\right)/|S| + max\left(x^i_{00}, x^i_{01}\right)/|S| \end{aligned}$$

Since the maximum is the average plus half the absolute difference, this is equal to

$$\begin{aligned} MG &= I(parent) - 1 + (x^i_{00} + x^i_{01} + x^i_{10} + x^i_{11})/2|S| + |x^i_{00} - x^i_{01}|/2|S| + |x^i_{10} - x^i_{11}|/2|S| \\ &= I(parent) - 1/2 + [|x^i_{00} - x^i_{01}| + |x^i_{10} - x^i_{11}|]/2|S| \end{aligned}$$

Choosing the attribute with the highest gain is equivalent to maximizing the misclassification gain. As seen above, maximizing this quantity is the same as maximizing $|x^i_{00} - x^i_{01}| + |x^i_{10} - x^i_{11}|$. Thus, according to the misclassification gain function, the best attribute to split is $A^{best} = \arg\max_{i \in [1,d]} |x^i_{00} - x^i_{01}| + |x^i_{10} - x^i_{11}|$. Note that if $A^i = A^{best}$ then for any $A^j \neq A^i$ $C^{i,j} = |x^i_{00} - x^i_{01}| + |x^i_{10} - x^i_{11}| - \left|x^j_{00} - x^j_{01}\right| - \left|x^j_{10} - x^j_{11}\right|$ is either zero or more. A different derivation of this quantity is given in Appendix B for any two arbitrary categorical attributes.

In a distributed setup, S is partitioned into n sets S_1 through S_n. $X^i_k = \begin{matrix} x^i_{k,00} & x^i_{k,01} \\ x^i_{k,10} & x^i_{k,11} \end{matrix}$ would therefore denote the cross table of attribute A^i and the class in the example set S_k. Note that $x^i_{00} = \sum_{k=1...n} x^i_{k,00}$ and $C^{i,j} = \left|\sum_{k=1...n} \left[x^i_{k,00} - x^i_{k,01}\right]\right| + \left|\sum_{k=1...n} \left[x^i_{k,10} - x^i_{k,11}\right]\right| - \left|\sum_{k=1...n} \left[x^j_{k,00} - x^j_{k,01}\right]\right| - \left|\sum_{k=1...n} \left[x^j_{k,10} - x^j_{k,11}\right]\right|$. Also, notice that $C^{i,j}$ is not, in general, equal to $\sum_{k=1...n} \left|x^i_{k,00} - x^i_{k,01}\right| + \sum_{k=1...n} \left|x^i_{k,10} - x^i_{k,11}\right| - \sum_{k=1...n} \left|x^j_{k,00} - x^j_{k,01}\right| - \sum_{k=1...n} \left|x^j_{k,10} - x^j_{k,11}\right|$. Still, using the indicators s^i_0 and s^i_1 defined above we can write $C^{i,j} = s^i_0 \sum_{k=1...n} \left[x^i_{k,00} - x^i_{k,01}\right] + s^i_1 \sum_{k=1...n} \left[x^i_{k,10} - x^i_{k,11}\right] - s^j_0 \sum_{k=1...n} \left[x^j_{k,00} - x^j_{k,01}\right] - s^j_1 \sum_{k=1...n} \left[x^j_{k,10} - x^j_{k,11}\right]$ which can be rewritten as

$$C^{i,j} = \sum_{k=1...n} \left(s^i_0 \left[x^i_{k,00} - x^i_{k,01}\right] + s^i_1 \left[x^i_{k,10} - x^i_{k,11}\right] - s^j_0 \left[x^j_{k,00} - x^j_{k,01}\right] - s^j_1 \left[x^j_{k,10} - x^j_{k,11}\right]\right);$$

This last expression, in turn, is simply a sum — across all peers — of a number $\delta^{i,j}_k = s^i_0 \left[x^i_{k,00} - x^i_{k,01}\right] + s^i_1 \left[x^i_{k,10} - x^i_{k,11}\right] - s^j_0 \left[x^j_{k,00} - x^j_{k,01}\right] - s^j_1 \left[x^j_{k,10} - x^j_{k,11}\right]$ which can be computed independently by each peer, assuming it knows the values of the indicators.

Finally, denote $\delta_k^{i,j}|abcd$ the value of $\delta_k^{i,j}$ assuming $s_0^i = a$, $s_1^i = b$, $s_0^j = c$, and $s_1^j = d$. Notice that, unlike $\delta_k^{i,j}$, $\delta_k^{i,j}|abcd$ can be computed independently by every peer, regardless of the actual values of s_0^i, s_1^i, s_0^j, and s_1^j.

It is therefore possible to compute A^{best} by concurrently running the following set of majority votes: two per attribute A^i, with inputs $x_{00}^i - x_{01}^i$ and $x_{10}^i - x_{11}^i$, to compute the values of s_0^i and s_1^i; and one for every per of attributes and every possible combination of s_0^i, s_1^i, s_0^j, and s_1^j. Given the results for s_0^i and s_1^i, one could select the right combination and ignore the other. Then, given all of the selected votes, one could find the attribute whose gain is higher than that of any other attribute. Below, we describe an algorithm which performs the same computation far more efficiently.

P2P Misclassification Minimization The P2P Misclassification Minimization *P*^{2}*MM* algorithm, Algorithm 11, aims to solve two problems concurrently: it will decide which of the attributes A^1 through A^d is A^{best} and at the same time compute the true value of s_0^i and s_1^i. The general framework of the *P*^{2}*MM* algorithm is that of pivoting: a certain A^i is selected as the potential A^{best}. Then the algorithm tries to validate this selection. If A^i turns out to be indeed the best attribute, then this is reported. On the other hand, the revocation of the validation proves that there must exist another attribute (or attributes) A^j which is (are) better than A^{best}. At this point, the algorithm again renames one of these better attributes A^j to be A^{best}, and tries to validate this by comparing it to the other attributes that turned out to be better than the previous selection i.e. previously suspected best. To provide the input to those comparisons, each peer computes $\delta_k^{i,j}|abcd$ relying on the current ad-hoc value of s_0^i, s_1^i, s_0^j, and s_1^j. The ad-hoc values peer k computes for s_0^i and s_1^i will be denoted by $s_{k,0}^i$ and $s_{k,1}^i$, respectively. To make sure those ad hoc results converge to the correct value, two additional majority votes are carried per attribute concurrent to those of the pivoting; in these, the inputs of peer k are $x_{00}^i - x_{01}^i$ and $x_{10}^i - x_{11}^i$, respectively.

Figure 4.1 shows the pivoting step as it advances. In the figure, there are five attributes

A_1, A_2, A_3, A_4 and A_5. The height of the bars denote their gains. In the first snapshot (first row), A_1 is selected as the pivot. A_1 is then compared to all the other attributes A_2, A_3, A_4 and A_5. All the attributes better than A_1 are reported in the third step — A_2, A_4 and A_5. In the fourth step, A_2 is selected as the pivot and compared to A_4 and A_5. Finally A_2 is output as the best attribute.

The P^2*MM* algorithm works in streaming mode: Every peer k takes two inputs — the set of its neighbors Γ_k and a set S_k of learning examples. Those inputs may (and often do) change over time and the algorithm responds to every such change by adjusting its output and by possibly sending messages. Similarly, messages stream in to the peer and, can influence both the output and the outgoing messages. The output of peer k is the attribute with the highest misclassification gain. This output, by nature, ad-hoc and may change in response to any of the events described above.

P^2*MM* is based on a large number of instances of the distributed majority voting algorithm described earlier in Section 4.3.2. On initialization, P^2*MM* invokes two instances of majority voting per attribute to determine the values of s_0^i and s_1^i — let M_0^i and M_1^i denote these majority votes. For each peer k, its inputs to these votes (instances) are $M_0^i.\delta_k = x_{k,00}^i - x_{k,01}^i$ and $M_1^i.\delta_k = x_{k,10}^i - x_{k,11}^i$. Additionally, for every pair $i < j \in [1 \dots d]$ P^2*MM* initializes sixteen instances of majority voting — one for each possible combination of values for s_0^i, s_1^i, s_0^j, and s_1^j. Those instances are denoted by $M_{abcd}^{i,j}$ with $abcd$ referring to the combination of values for s_0^i, s_1^i, s_0^j, and s_1^j. For each peer k, its inputs to these instances are $M_{abcd}^{i,j}.\delta_k = a\left[x_{k,00}^i - x_{k,01}^i\right] + b\left[x_{k,10}^i - x_{k,11}^i\right] - c\left[x_{k,00}^j - x_{k,01}^j\right] - d\left[x_{k,10}^j - x_{k,11}^j\right]$. A related algorithm, distributed plurality voting (DPV) [116], could be used to describe our algorithm for selecting one of the sixteen comparisons $M_{abcd}^{i,j}|abcd$ when $abcd$ are static. But since in our case the values of s_0^i, s_1^i, s_0^j, and s_1^j are always changing, it seems difficult to follow the same description as given in [116].

Following initialization, the algorithm is event based. Each peer k is idle unless there

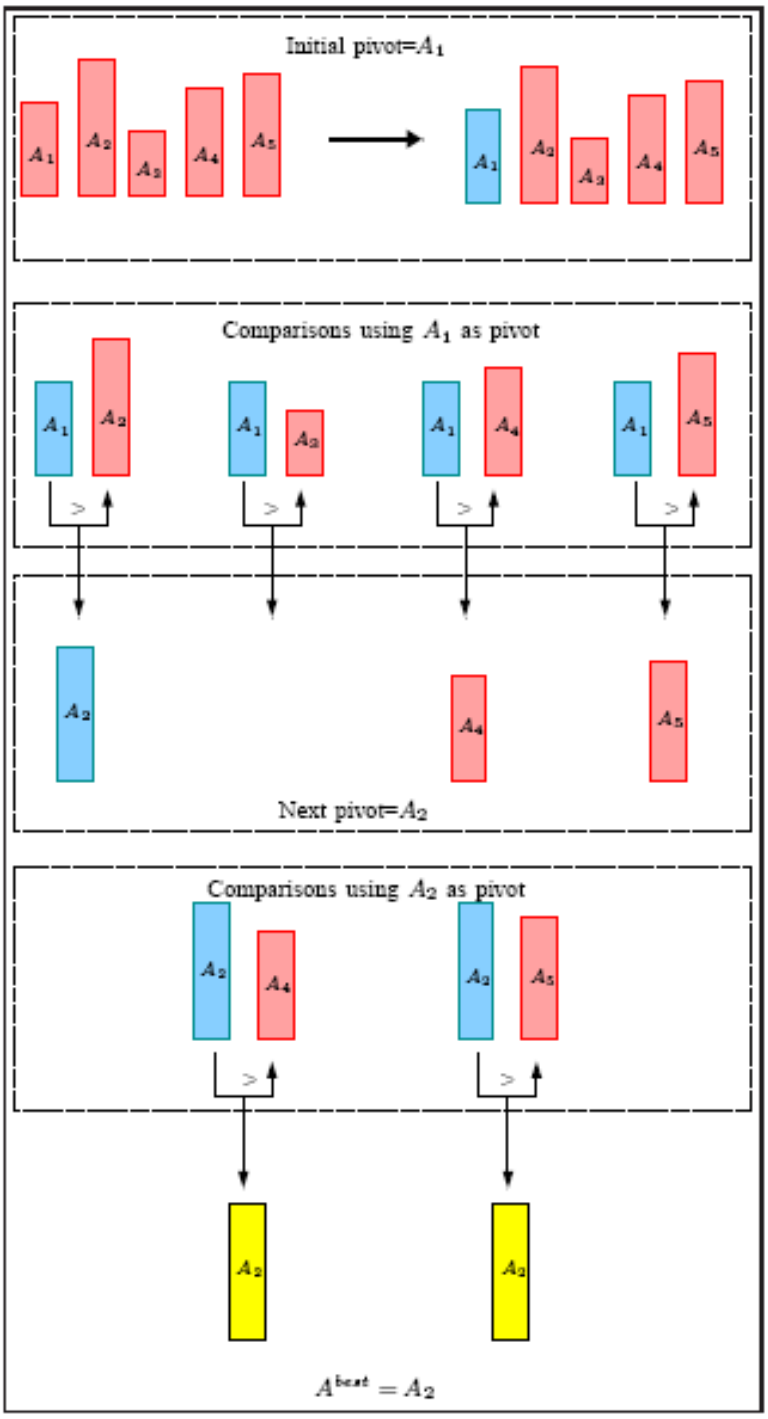


FIG. 4.1. Figure showing the pivot selection process. In the first row, A_1 is selected as the pivot. A_1 is then compared to all other attributes and all the ones better than A_1 are selected in the third snapshot. A_2 is selected as the pivot. Finally A_2 is output as the best attribute.

is a change in Γ_k or S_k, or an incoming message changes the Δ_k of one of the instances of majority voting. On such event, the simple solution would be to check the conditions for sending messages in any of the majority votes. However, as shown by [105][116] pivoting can be used to reduce the number of conditions checked from $O\left(d^2\right)$ to an expected $O(d)$. In DPV, the following heuristic is suggested for pivot selection: the pivot would be the vote which has the largest agreement value with any of the neighbors. It is proved that this method is deadlock-free and it is shown to be effective in reducing the number of conditions checked. We implement the same heuristic, choosing the pivot as the attribute A^i which has the largest $M^{j,i}.\Delta_{k,\ell}$ for $j < i$ or the smallest $M^{i,m}.\Delta_{k,\ell}$ for $m > i$ and for any neighbor ℓ. If the test for any $M^{i,j}$ fails, $M^{i,j}.\Delta_{k,\ell}$ needs to be modified by sending a message to ℓ. P^2*MM* does this by sending a message which will set $M^{i,j}.\Delta_{k,\ell}$ to $\alpha M^{i,j}.\Delta_k$ (α is set to $\frac{1}{2}$ by default), which is in line with the findings of [195].

Notice $M^{i,j}_{abcd}.\delta_k = -M^{i,j}_{-a-b-c-d}.\delta_k$ and thus half of the comparisons actually replicate the other half and can be avoided. This optimization is avoided in the pseudocode in order to maintain conciseness. Also notice that while the peer updates in the context of any of the majority votes $M^{i,j}_{abcd}$, it will only respond with a message for the majority vote $M^{i,j}$ — the instance with a, b, c, and d equal to $s^i_{k,0}$, $s^i_{k,1}$, $s^j_{k,0}$, and $s^j_{k,1}$, respectively. The rest of the instances are, in effect, 'suspended' and cause no communication overhead. Next we show the P^2MM algorithm is eventually correct.

Theorem 4.4.1. *Each peer running the P^2MM algorithm eventually converges to the correct pivot.*

Proof. To see why P^2*MM* convergence is guaranteed, first notice that eventual correctness of each of the majority votes M^i_0 and M^i_1 is guaranteed because the condition for sending messages is checked for every one of them each time the data changes or a message is received. Next, consider two neighbor peers who choose different pivots, peer k which selects i and peer ℓ which selects j. Since both k and ℓ will check the condition of $M^{i,j}$,

Input: Γ_k, S_k
Output: the attribute A^{pivot}

Initialization:
begin
  forall $A^i \in \left\{A^1 \dots A^d\right\}$ **do**
    Initialize two instances of LSD-Majority $\left(M_0^i, M_1^i\right)$ with *inputs* $x_{k,00}^i - x_{k,01}^i$ and $x_{k,10}^i - x_{k,11}^i$, *knowledge* $M_0^i.\Delta_k$ and $M_1^i.\Delta_k$, and *agreement* $M_0^i\Delta_{k,\ell}$ and $M_1^i\Delta_{k,\ell}$ respectively ($\forall \ell \in \Gamma_k$);
  end
  forall $[a, b, c, d \in \{-1, 1\}] \bigwedge \left[A^i, A^j \in \left\{A^1 \dots A^d\right\}\right]$ **do**
    Initialize an instance of LSD-Majority $\left(M_{abcd}^{i,j}\right)$ with *input* $\delta_k^{i,j}|abcd$, *knowledge* $M_{abcd}^{i,j}.\Delta_k$ and *agreement* $M_{abcd}^{i,j}\Delta_{k,\ell}$ ($\forall \ell \in \Gamma_k$), where $a = s_{k,0}^i$, $b = s_{k,1}^i$, $c = s_{k,0}^j$, and $d = s_{k,1}^j$;
  end
end
On any event:
begin
  forall $A^i \in \left\{A^1 \dots A^d\right\}$ *and every* $\ell \in \Gamma_k$ **do**
    if *not* $M_0^i.\Delta_k \leq M_0^i.\Delta_{k,\ell} < 0$ *and not* $M_0^i.\Delta_k \geq M_0^i.\Delta_{k,\ell} \geq 0$ **then** call **Send**$\left(M_0^i, \ell\right)$;
    if *not* $M_1^i.\Delta_k \leq M_1^i.\Delta_{k,\ell} < 0$ *and not* $M_1^i.\Delta_k \geq M_1^i.\Delta_{k,\ell} \geq 0$ **then** call **Send**$\left(M_1^i, \ell\right)$;
  end
end
repeat
  Let $pivot = \arg \max_{i \in [1 \dots d]} \left\{ \max_{\ell \in N_k, j<i, m>i} \left\{M^{j,i}.\Delta_{k,\ell} - M^{i,m}.\Delta_{k,\ell}\right\} \right\}$;
  forall $A^i \in \left\{A^1 \dots A^{pivot-1}\right\}$ *and every* $\ell \in \Gamma_k$ **do**
    if *not* $M^{i,pivot}.\Delta_k \leq M^{i,pivot}.\Delta_{k,\ell} < 0$ *and not* $M^{i,pivot}.\Delta_k \geq M^{i,pivot}.\Delta_{k,\ell} \geq 0$ **then** call **Send**$\left(M^{i,pivot}, \ell\right)$;
  end
  forall $A^i \in \left\{A^{pivot+1} \dots A^d\right\}$ *and every* $\ell \in \Gamma_k$ **do**
    if *not* $M^{pivot,i}.\Delta_k \leq M^{pivot,i}.\Delta_{k,\ell} < 0$ *and not* $M^{pivot,i}.\Delta_k \geq M^{pivot,i}.\Delta_{k,\ell} \geq 0$ **then** call **Send**$\left(M^{pivot,i}, \ell\right)$;
  end
until *while* $pivot$ *changes* ;
if MessageRecvd(ℓ, (id, δ)) **then**
  Let M be a majority voting instance with $M.id = id$;
  $M.\delta_{\ell,k} \leftarrow \delta$;
end
Function Send((M, ℓ)**)**
begin
  $M.\delta_{k,\ell} = \alpha M.\Delta_k + M.\delta_{\ell,k}$;
  Send to ℓ $(M.id, M.\delta_{k,\ell})$;
end

Algorithm 11: P2P Misclassification Minimization (P^2MM)

and since $M^{i,j}.\Delta_{k,\ell} = M^{i,j}.\Delta_{\ell,k}$ at least one of them would have to change its pivot. Thus, peers will continue to change their pivot until they all agree on the same pivot. To see that peers will converge to the decision that the pivot is the attribute with the highest gain (denote it A^m), assume they converge on another pivot. Since the condition of votes comparing the A^{pivot} to any other attribute is checked whenever the data changes, it is guarantee that if $m < pivot$ then $M^{m,pivot}.\Delta_{k,\ell}$ will eventually be larger than zero for all k and $\ell \in \Gamma_k$ and if $pivot < m$ then $M^{m,pivot}.\Delta_{k,\ell} > 0$ for all k and $\ell \in \Gamma_k$. Thus, the algorithm will replace the pivot with either m or another attribute, but would not be able to converge on the current pivot. This completes the proof of the correct convergence of the P^2*MM* algorithm. □

Complexity The P^2*MM* algorithm compares the attributes in an asynchronous fashion and outputs the best attribute. Consider the case of comparing only two attributes. The worst case communication complexity of the P^2*MM* algorithm is O(*size of the network*). This can happen when the misclassification gains of the two attributes are very close. Since our algorithm is eventually correct, the data will need to be propagated through the entire network i.e. all the peers will need to communicate to find the correct answer. Thus the overall communication complexity of the P^2*MM* algorithm, in the worst case, is O(*size of the network*). Now if the misclassification gains of the two attributes are not very close (which is often the case for most datasets), the algorithm is not global; rather the P^2*MM* algorithm can prune many messages as shown by our extensive experimental results. Finally formulating the complexity of such data dependent algorithms in terms of the complexity of the data is a big open research issue even for simple primitives such as majority voting protocol [196], leave aside the complex P^2*MM* algorithm presented here.

Figure 4.2 demonstrates how two attributes A_i and A_j are compared by two peers P_k and P_ℓ. At time t_0, the peers initialize all the sixteen votes $M^{i,j}_{abcd}$ along with the votes for the indicator variables. At time t_1, the values of the indicator variables are $\{1, 1, 1, -1\}$ for

P_k. Hence it only sends messages corresponding to the vote $M^{i,j}_{111-1}$. All the rest fifteen votes are in a suspended state. Similarly the figure shows the suspended votes for P_ℓ at time t_2. Finally it depicts that at time t_4, the peers converge to the same vote which is not suspended viz. $M^{i,j}_{1-11-1}$.

Figure 4.3 shows the pivot selection process for three attributes A_h, A_i and A_j for peer P_k having two neighbors P_ℓ and P_m. Snapshot 1 shows the knowledge (Δ_k) and agreements of P_k ($\Delta_{k,\ell}$ and $\Delta_{k,m}$) for the three attributes. Since pivot is the highest agreement, A_h is selected as the pivot. Now there is a disagreement between Δ_k and $\Delta_{k,\ell}$ for A_h. This results in a message and subsequent reevaluation of $\Delta_{k,\ell}$ (to be set equal to Δ_k). In the next snapshot, A_i is selected as the pivot and since $\Delta_{k,\ell} < \Delta_k$, no message needs to be sent.

Comment: The *P²MM* algorithm above utilizes the assumption that the data at all peers in boolean (attributes and class labels). The boolean attributes assumption can be relaxed to arbitrary categorial data at the expense of increased majority voting instances per attribute pair (the number of instances increases exponentially with the number of distinct attribute values). Another approach to relaxing the boolean attributes assumption could be to treat each attribute distinct value as its own boolean attribute. As a result, each categorical attribute with v distinct values is treated as v boolean attributes. Here, the number of majority voting instances per pair of attributes increases only linearly with the number of distinct attribute values. However, the issue of deciding which attribute has highest misclassification gain on the basis of the associated boolean attributes is not entirely clear and is the subject of more research. In Appendix B we show how our framework can be easily extended for arbitrarily categorical attributes.

4.4.2 Speculative Decision Tree Induction

P²MM can be used to decide which attribute would best divide a given set of learning examples. It is well known that decision trees can be induced by recursively and greedily

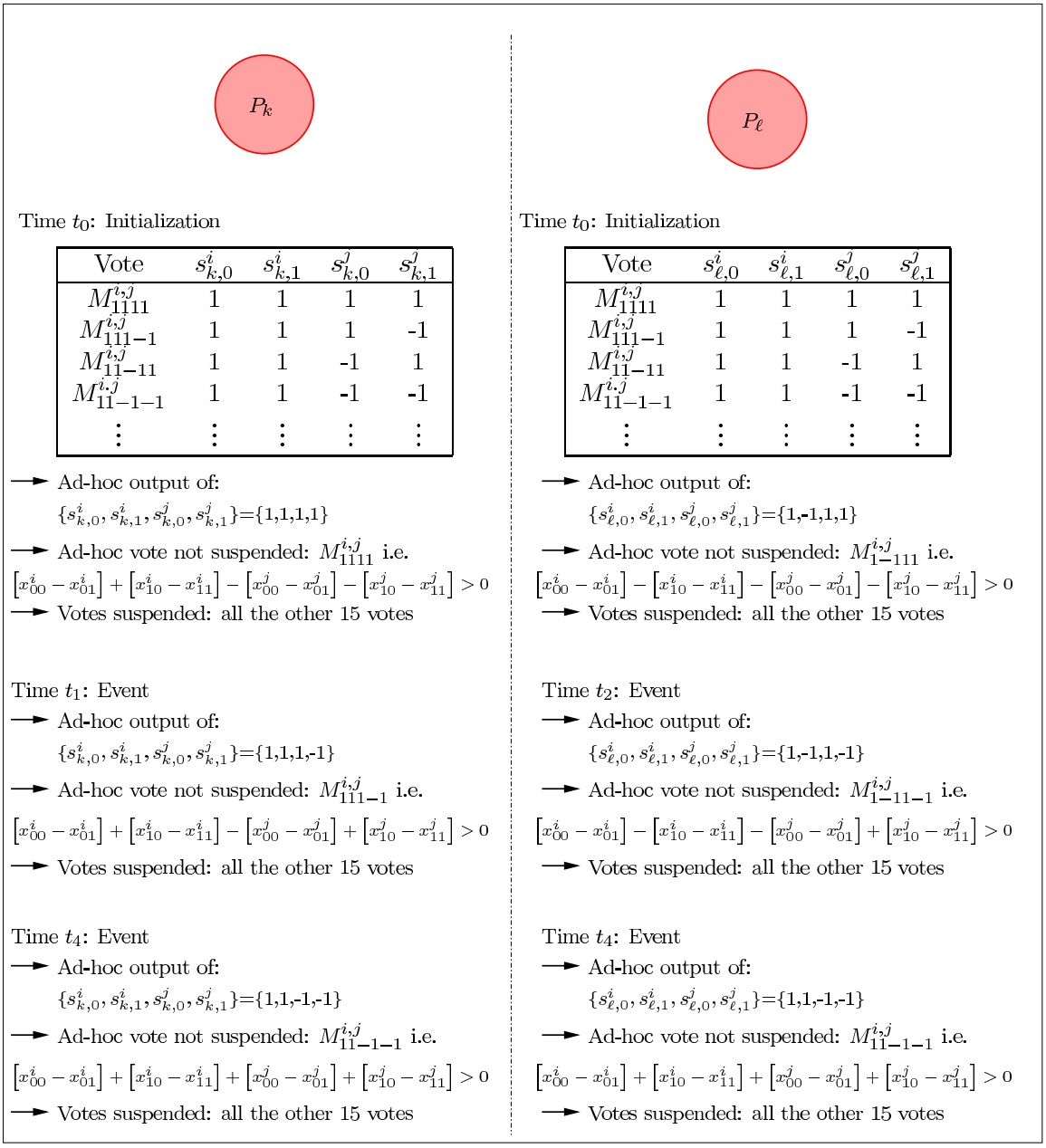


FIG. 4.2. Comparison of two attributes A_i and A_j for two peers P_k and P_ℓ. The figure also shows some example values of the indicator variables and the suspended/not suspended votes in each case.

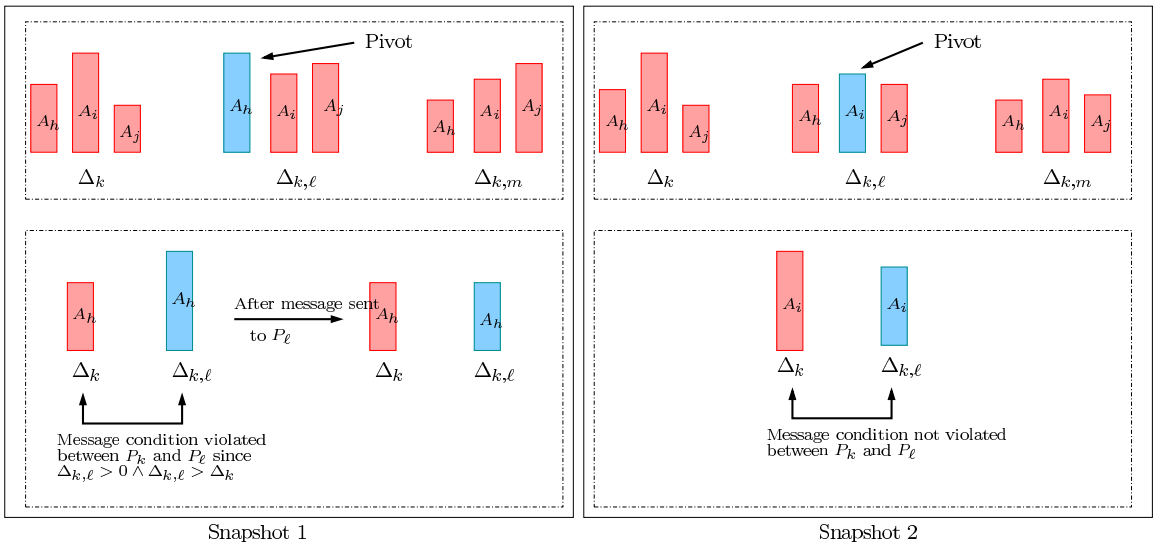


FIG. 4.3. The pivot selection process and how the best attribute is selected. The pivot is shown at each round.

dividing a set of learning examples — starting from the root and proceeding onwards with every node of the tree (e.g., ID3 and C4.5 algorithms [154][155]). In a P2P set-up the progression of the algorithm needs to be coordinated among all peers, or they might end up developing different trees. In smaller scale distributed systems, occasional synchronization usually addresses coordination. However, since a P2P system is too large to synchronize, we prefer speculation [98].

The starting point of tree development — the root — is known to all peers. Thus, they can all initialize P^2MM to find out which attribute best splits the example set of the root. However, the peers are only guaranteed to converge to the same attribute selection eventually and may well choose different attributes intermediately. Several questions thus arise: How and when should the algorithm commit resources to a specific split of a node, what should be done if such split appears to be wrong after resources were committed and what should be done about incoherence between neighboring peers?

The P2P Decision Tree Induction (P^2DTI, see Alg. 12) algorithm has two main func-

tionalities. Firstly, it manages the ad hoc solution which is a decision tree composed of *active* nodes. The root is always active and so is any node whose parent is active provided that the node corresponds with one of the values of the attribute which best splits its parent's examples — i.e., the ad hoc solution of P^2*MM* as computed by the parent. The rest of the nodes are *inactive*. A node (or a whole subtree) can become inactive because its parent (or fore-parent) have changed its preference for splitting attribute. Inactive nodes are not discarded; a peer may well accept messages intended for inactive nodes — either because a neighbor considers them active or because the message was delayed by the network. Such a message would still update the majority voting to which it is intended. However, peers never send messages resulting from an inactive node. Instead, they check, whenever a peer becomes active, whether there are pending messages (i.e., majority votes whose test require sending messages) and if so they send those messages.

Another activity which occurs in active nodes is further development of the tree. Each time a leaf it is generated it is inserted into a queue. Once every τ time units, the peer takes the first active leaf in the queue and develops it according to the ad hoc result of P^2*MM* for that leaf. Inactive leaves which precede this leaf in the queue are re-inserted at the end of the queue.

Last, it may happen that a peer receives a message in the context of a node it had note yet developed. Such messages are stored in the *out-of-context* queue. Whenever a new leaf is created, the out-of-context queue is searched and messages pertaining to the new leaf are processed.

A high level overview of the speculative decision tree induction process is shown in Figure 4.4. Filled rectangles represent newly created nodes. The first snapshot shows the creation of the root with A_1 as the best attribute. The root is split in the next snapshot, followed by further development of the left path. The fourth snapshot shows how the root is changed to a new attribute A_2 and the entire tree rooted at A_1 is made inactive (yellow

Input: S – a set of learning examples, τ – mitigation delay
Output: ad-hoc decision tree

Initialization:
begin
  Create a root and let $root.S \leftarrow S$; $node \leftarrow \{root\}$;
  Push $root$ to $queue$; Send BRANCH message to self with delay τ;
end
On BRANCH message:
begin
  Send BRANCH message to self with delay τ;
  for ($i \leftarrow 0$, $\ell \leftarrow null$; $i < queue.length$ *and not active*(ℓ); i++) **do**
    Pop head of queue into ℓ;
    if *not active*(ℓ) **then** enqueue ℓ;
    if *active*(ℓ) **then** $A^j \leftarrow$ ad-hoc solution of P^2*MM* for ℓ; call **Branch**(ℓ, j);
  end
end
if DataMsgReceived$(n, data)$ **then**
  if $n \notin nodes$ **then** store $\langle n, data \rangle$ in $out - of - context$;
  else Transfer the $data$ to the P^2*MM* instance of n;
  if *active*(n) **then Process**(n);
end
Function Active (n)
begin
  if $n = null$ *or* $n = root$ **then** return true;
  $A^j \leftarrow$ ad-hoc solution for P^2*MM* for $n.parent$;
  if $n \notin n.parent.sons\,[j]$ **then** return false;
  Return Active$(n.parent)$;
end
Function Process(n)
begin
  Perform tests required by P^2*MM* for n and send any resulting messages;
  $A^j \leftarrow$ ad-hoc solution for P^2*MM* for n;
  if $n.sons\,[j]$ *is not empty* **then for** *each* $m \in n.sons\,[j]$ **do** call Process(m);
  else push n to the tail of the queue;
end
Function Branch$(\ell, , j)$
begin
  Create two new leaves ℓ_0 and ℓ_1;
  $\ell_0.parent \leftarrow \ell$ and $\ell_1.parent \leftarrow \ell$;
  Set $\ell_0.S \leftarrow \{s \in \ell.S : s\,[j] = 0\}$ and $\ell_1.S \leftarrow \{s \in \ell.S : s\,[j] = 1\}$;
  Deliver the messages from $out - of - ccontext$ to ℓ_0 and ℓ_1;
  $\ell.sons\,[j] = \{\ell_0, \ell_1\}$; add ℓ_0, ℓ_1 to nodes; push ℓ_0 and ℓ_1 to the tail of the queue;
end

Algorithm 12: P2P Decision Tree Induction (P^2DTI)

part). Finally as time progresses, the tree rooted at A_2 is further developed. If it so happens that A_1 now becomes better than A_2, the old inactive tree will now become active and the tree rooted at A_2 will become inactive.

4.4.3 Stopping Rule Computation

A decision tree induction algorithm cannot be considered complete without a pruning technique stating which branches over-fit the data. Pruning techniques can be divided between *post*-pruning — removing nodes of whole subtrees after the tree is induced — and *pre*-pruning — instructing the algorithm which nodes not to develop in the first place. Since the P^2*DTI* algorithm works on streaming data the idea of first inducing the tree and afterwards pruning it seems less suitable (because there is no specific point in time in which pruning should begin). We therefore focus on pre-pruning heuristics. Several common pre-pruning techniques can easily be adopted by P^2*DTI*, because they use simple statistics. We will describe three such techniques.

The simplest pre-pruning technique is to terminate new leave development which the tree reaches a pre-specified depth. The biggest benefit of this technique is that it limits the resources used by the algorithm. This technique is trivial to implement as part of P^2*DTI* and performs quite favorably in experiments. It's greatest disadvantage is that the depth limit has to be found in trial and error, and is the same for all leaves.

A second popular pre-pruning technique is to require a minimal degree of gain from every split in the tree, or abort the split if the gain is below the required number. Splitting any given nodes according to an attribute A^i will have non-negative gain if $|x^i_{00} + x^i_{10} - x^i_{01} - x^i_{11}| - |x^i_{00} - x^i_{01}| - |x^i_{10} - x^i_{11}| \geq 0$. Let $s^i = sign\left(x^i_{00} + x^i_{10} - x^i_{01} - x^i_{11}\right)$, the previous formula can be rewritten $s^i\left(x^i_{00} + x^i_{10} - x^i_{01} - x^i_{11}\right) - s^i_0\left(x^i_{00} - x^i_{01}\right) - s^i_1\left(x^i_{10} - x^i_{11}\right) \geq 0$. Just as in Section 4.4.1, this formula can be evaluated by holding eight different majority votes per attribute, one for every possible value of s^i, s^i_0, and s^i_1,

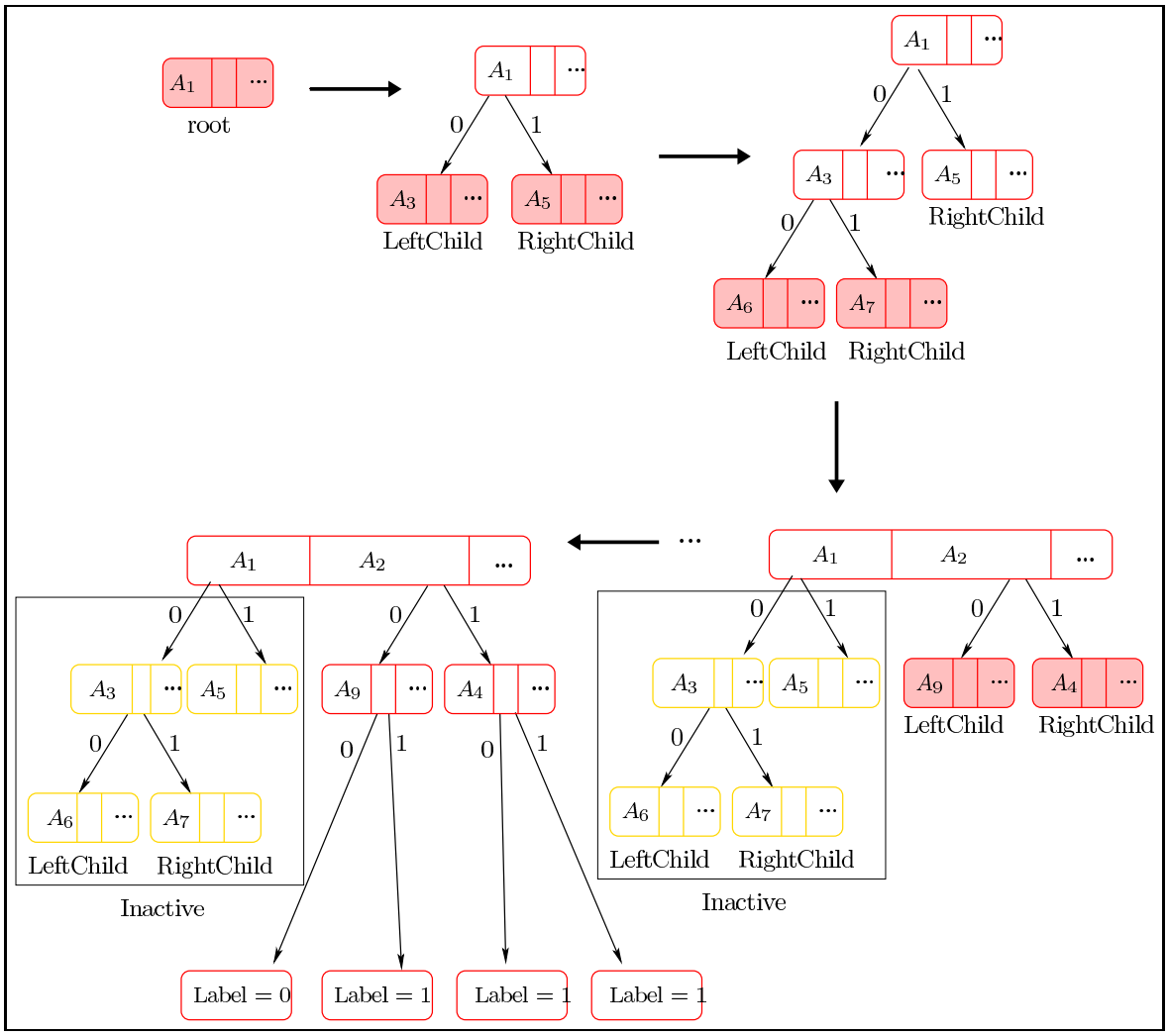


FIG. 4.4. Figure showing how the speculative decision tree is build by a peer. Filled rectangles represent the newly created nodes. In the first snapshot the root is just created with A_1 as the current best attribute. The root is split into two children in the second snapshot. The third snapshot shows further development of the tree by splitting the left child. In the fourth snapshot, the peer gets convinced that A_2 is the best attribute corresponding to the root. Earlier tree is made inactive and a new tree is developed with split at A_2. Fifth snapshot shows the leaf label assignments.

and then selecting one of them according to the ad hoc result of separate votes on the values of s^i, s^i_0, and s^i_1. Notice that votes for s^i_0 and s^i_1 are held anyhow. Also, the input for the vote for s^i is independent of i so just one extra vote is needed for all A^i. Furthermore, for all nodes except the root, a vote on the value of s^i is actually held in the context of the parent of the node.

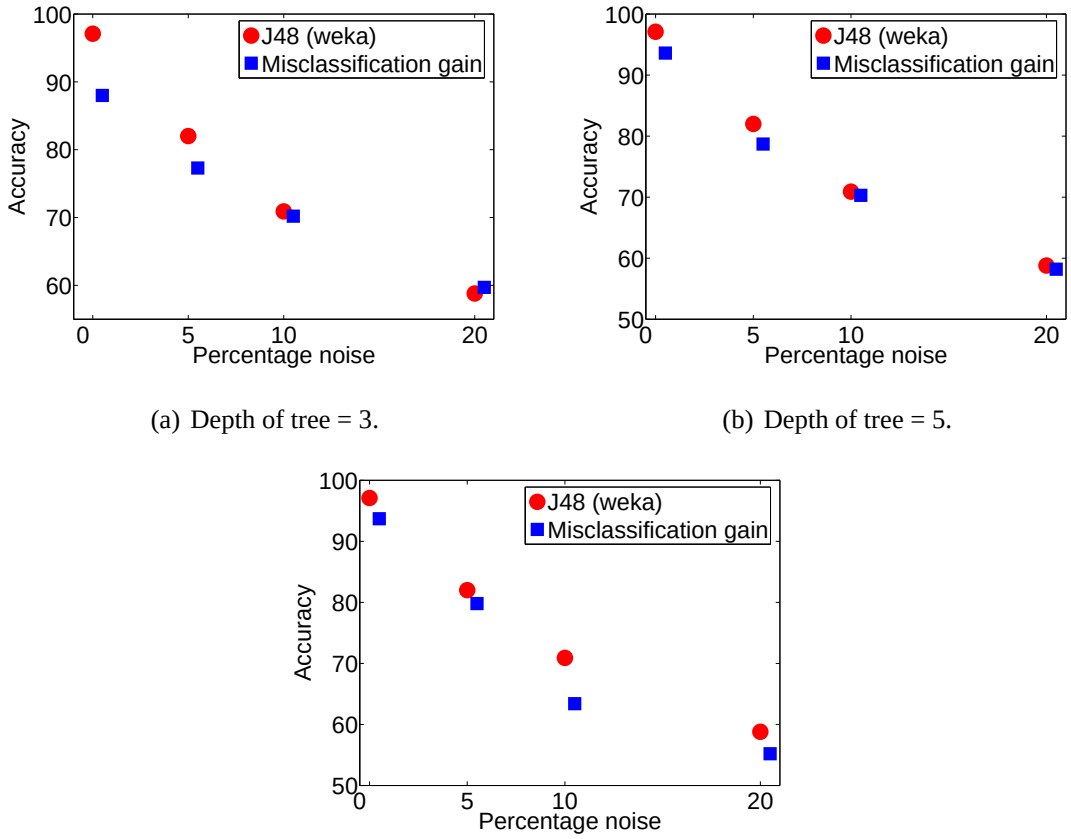
A third pruning technique is to require non-negative gain from every split when this gain is measured on a test set rather than on the learning set. This method introduces little complexity beyond the described above: the difference is that each node, starting with the root, should be associated with an additional set of examples and that the input to the votes regarding pruning be taken from that set. Notice that the input to votes of the s^i type should still be taken from the learning set.

In the next section we present a comparative study of the accuracy of a decision tree algorithm using misclassification gain as the impurity measure with fixed depth stopping and a standard entropy-based pruned decision tree J48 implemented in weka [194].

4.4.4 Accuracy on Centralized Dataset

The proposed distributed decision tree algorithm P^2*DTI* deviates from the standard decision tree induction algorithm in two ways: (1) instead of using entropy or gini-index as the splitting criteria, we have used misclassification gain, and (2) as a stopping rule, we have limited the depth of our trees (which effects the communication complexity of our distributed algorithm as we show later).

In this section we report the results of the comparison of a decision tree induced using misclassification gain as the impurity measure and fixed depth stopping rule to that of an off-the-shelf entropy-based pruned decision tree J48 implemented in Weka [194] on a centralized dataset. There are 500 tuples and 10 attributes in the centralized dataset generated using the scheme discussed in Section 3.9.1. We have varied the noise in the



(a) Depth of tree = 3.

(b) Depth of tree = 5.

(c) Depth of tree = 7.

FIG. 4.5. Comparison of accuracy (using 10-fold cross validation) of J48 weka tree and a tree induced using misclassification gain with fixed depth stopping rule on a centralized dataset. The three graphs correspond to depths of 3, 5 and 7 of the misclassification gain decision tree.

data ranging from 0% to 20%. For both the trees, the accuracy is measured using 10-fold cross validation. Figure 4.5 presents the comparative study. The circles correspond to the accuracy of J48 and the squares correspond to the accuracy of the misclassification gain decision tree. These set of results point out some important facts:

1. In most cases the misclassification gain decision tree results in a loss of accuracy over J48. This is not unexpected due to the restrictive nature of the decision tree induction algorithm employed. But the accuracy loss is modest. Moreover, due to the heavy communication and synchronization cost of centralizing and applying J48, this modest loss of accuracy seems quite reasonable.

2. The decrease in accuracy of the misclassification gain decision tree when going to depth 7 at high noise levels is likely due to over-fitting; since for a depth of 3 the average number of tuples per leaf is 56 compared to only 3 tuples per leaf for depth of 7.

3. The P^2DTI algorithm is guaranteed to converge to the misclassification gain decision tree with fixed stopping depth. Therefore, once convergence is reached, P^2DTI might loose some accuracy with respect to J48 and this is quantified by the accuracy of the misclassification gain decision tree with fixed stopping depth as shown here.

Since limiting the depth affects the communication complexity of our P^2DTI algorithm, we will use depths of 3, as it produces quite accurate trees for all noise levels.

4.5 Experiments

To validate the performance of our decision tree algorithm, we conducted experiments on a simulated network of peers. In this section we discuss the experimental setup, measurement metric and the performance of the algorithm.

4.5.1 Experimental Setup

Our implementation makes use of the Distributed Data Mining Toolkit (DDMT) [44] which is a JADE-LEAP based event simulator developed by the DIADIC research lab at UMBC. The topology is generated using BRITE [23] — an open software for generating network topologies. We have used the *Barabasi Albert (BA)* model in BRITE since it is often considered a reasonable model for the Internet. We use the edge delays defined in BA as the basis for our time measurement[2]. On top of each network generated by BRITE, we overlayed a communication tree.

4.5.2 Data Generation

The input data of a peer is generated using the scheme proposed by Domingos and Hulten [48]. Each input data point is a vector in $\{0,1\}^d \times \{0,1\}$. The data generator is a random tree built as follows. At each level of the tree, an attribute is selected randomly and made an internal node of the tree with the only restriction that attributes are not repeated along the same path. After the tree is built up to a depth of 3, a node is randomly made a leaf with a probability of p along with a randomly generated label. We limit the depth of the tree to maximum 6, and make all the non-leaf nodes a leaf (with random labels) after it exceeds that depth. Whenever a peer needs an additional point, it generates a random vector in the d-dimensional space and then passes it through the tree. The label it gets assigned to forms the class label for that input vector. This forms noise-free input vectors. In order to add noise, the bits of the vectors (including the class label) are flipped with a certain probability. Therefore, $n\%$ noise means that each bit of the input vector is flipped with $n\%$ chance and the new value of that bit is chosen uniformly from all the possibilities (including the original value). The data generator is changed every 5×10^5 simulator ticks, thereby creating an epoch. A typical experiment consists of 10 equal length epochs In addition, throughout the

[2]Wall time is meaningless when simulating thousands of computers on a single PC.

experiment we change 10% data of each peer after every 1000 clock ticks. Therefore, in all our experiments there are two levels of data change — (1) stationary change when we sample from the same data distribution every 1000 simulator ticks, and (2) dynamic change when the data distribution changes after every 5×10^5 simulator ticks.

4.5.3 Measurement Metric

The two measurements of our algorithm are the *quality* of the result and the *cost* incurred.

Given a test dataset to each peer, generated from the same distribution as the local dataset, quality is measured in terms of the percentage of correctly classified tuples of this test set. We report both the *stationary accuracy* which refers to the accuracy measured during the last 80% of the epochs (and hence correspond to stationary changes) and the *overall accuracy*. Each quality graph in Figures 4.6, 4.7, 4.8, 4.9, 4.10 and 4.11—4.13 reports two quantities — (1) the average quality over all peers, all epochs and 10 independent trials (the center markers) and (2) the standard deviation over 10 independent trials (error bars).

For measuring the cost of the algorithm we report two quantities — *normalized messages* sent and *normalized bytes* transferred. Our measurement metric for the normalized messages is the number of messages sent by each peer per unit of leaky bucket L. For an algorithm whose communication model is broadcast, its normalized messages is 2, considering 2 neighbors on an average per peer. We report both the overall messages and the monitoring messages; the latter refers to the "wasted effort" of the algorithm. For a given time instance, if a peer needs to send k separate messages corresponding to different majority votes to one particular peer, it is counted as one message to that neighbor.

Similarly, to understand the actual communication overhead of our algorithm in terms of the number of bytes sent, we report both the *overall* and *monitoring* bytes transferred, per unit of L. In every raw message the distributed algorithm sends 5 numbers — the data

of the vote, the id of the vote, the id of the attributes which this vote corresponds to, the path of the tree and the maximum pivot. Considering the above example, the number of bytes sent is $5 * k$ per neighbor. As before, for a broadcast based algorithm, having an attribute cross-table with four entries (2 values and 2 classes), its bytes sent would be *no of attributes* $\times 4 \times 2$. The factor of two is assuming 2 neighbors per peer. For example, with 10 attributes, the number of bytes sent per L is 80. Similar to what we did for quality, we have plotted both the average cost and the standard deviation of the result over 10 independent trials.

There are three parameters of the P^2*DTI* algorithm that we have explored — (1) the number of local tuples or the size of the local dataset $|S_i|$, (2) the depth of the induced tree, and (3) the size of the leaky bucket L. The measurement points for the local data points per peer are 250, 500, 1000, 2000 and 4000. For the depth of tree, we used values of 2, 3, 4, 5 and 7 while we varied L among 1000, 2000, 3000 and 4000. The values of L are in simulator ticks where the average edge delay is about 1100 time units.

The data generator had two parameters — (1) noise in the data varied between 0%, 5%, 10% and 20%, and (2) number of attributes (10, 15, and 20).

Finally, as a system parameter we varied the number of peers from 50 to 1500.

Unless otherwise stated, we have used the following default values: $|S_i| = 500$, depth of the tree = 3, noise = 10%, number of attributes = 10, number of peers = 1000, and $L = 1000$ (where the average edge delay is about 1100 time units). Under these values, for a broadcast algorithm, the number of normalized messages is 2 while the number of normalized bytes is $10 \times 4 \times 2 = 80$.

In all our experiments we have observed the following phenomenon. As soon as the epoch changes, the accuracy of the algorithm goes down and the communication increases since the algorithm adapts to the new distribution. As soon as the distribution becomes stable, the message overhead reduces and the accuracy improves. To take note of this, we

have plotted both the overall and stationary behavior of the algorithm with respect to the different parameters.

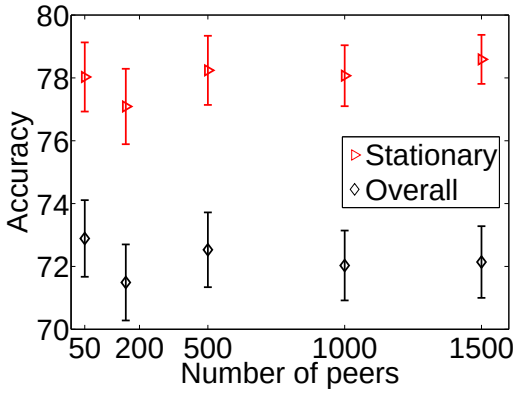
In the next few sections we present the performance of the P^2*DTI* algorithm on the different parameters.

4.5.4 Scalability

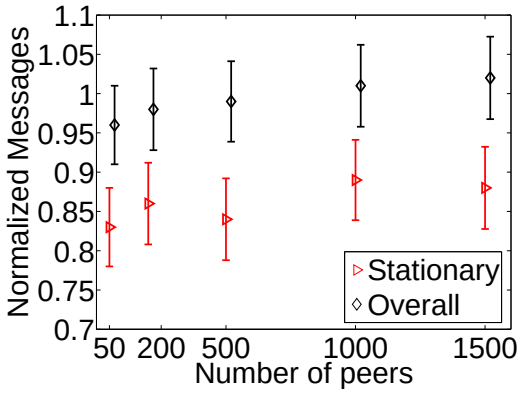
Our first set of results demonstrate the scalability of the P^2*DTI* algorithm as the number of peers is varied from 50 to 1500. The number of peers has no effect on the performance as we see in Figure 4.6. In Figure 4.6(a), both the overall and stationary accuracy converges to a constant as the number of peers is increased. Normalized messages and normalized bytes transferred, as shown in Figures 4.6(b) and 4.6(c), changes very slowly and almost remains a constant as the number of peers is increased. Since our algorithm relies on some data dependent rules to prune messages, the total number of peers has little effect on the quality or the cost. Hence the algorithm is highly scalable. Note that for an algorithm which broadcasts sufficient statistics to maintain the trees the normalized messages and normalized bytes transferred would be 2 and 80 respectively. Our results show a significant improvement.

4.5.5 Data Tuples per Peer

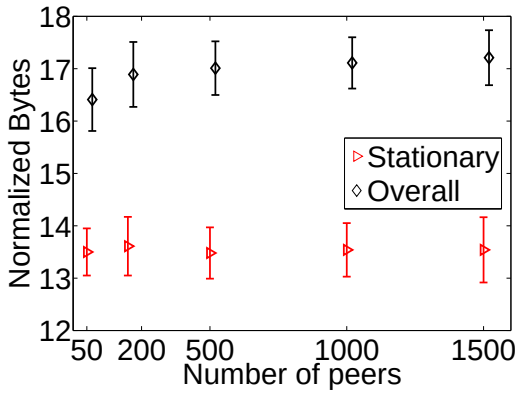
In this section we have experimented with the first algorithm parameter — the number of tuples in the local dataset $|S_i|$. Figure 4.7 summarizes the results. Stationary accuracy increases from 72% to 85%, stationary messages decrease from 1.21 to 0.32 and stationary bytes reduce from 20.9 to 4.24 as the size of the local dataset is increased from 250 tuples per peer to 4000 tuples per peer. This is true since with increasing $|S_i|$, the global tree is induced on a larger dataset, leading to better accuracy. Moreover, with increasing $|S_i|$, the algorithm can capture more variability in the distribution (since the majority votes are run



(a) Quality vs. number of peers.



(b) Normalized messages vs. number of peers.



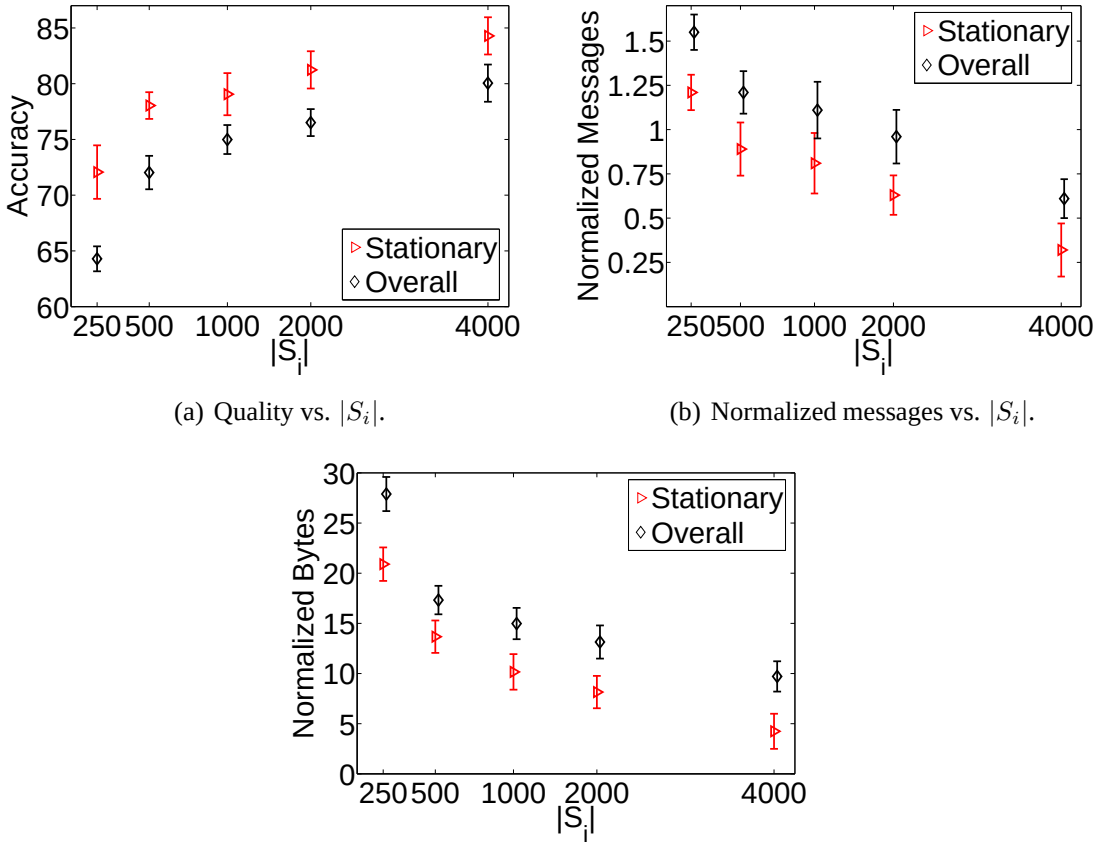
(c) Normalized bytes transferred vs. number of peers.

FIG. 4.6. Dependence of the quality and cost of the decision tree algorithm on the number of peers. The accuracy and the cost is almost invariant of the number of peers.

on a larger dataset) leading to lower communication. Even for the smallest dataset size of 250, the normalized messages is 1.21 and the stationary bytes is 20.9, both are far less than 2 and 80 respectively considering the broadcast algorithm.

4.5.6 Depth of Tree

As pointed out in Section 4.4.4, depth of the decision tree induced by the P^2DTI algorithm affects the cost of the algorithm. In this section, we validate this result. The



(a) Quality vs. $|S_i|$.

(b) Normalized messages vs. $|S_i|$.

(c) Normalized bytes transferred vs. $|S_i|$.

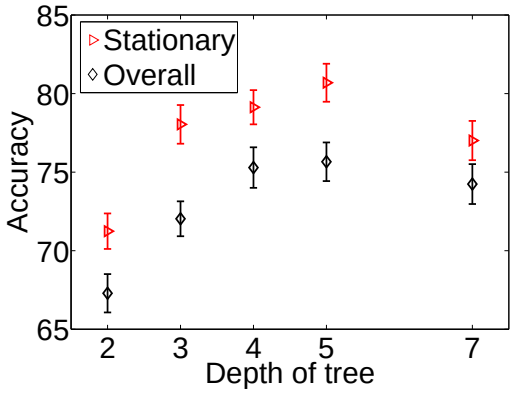
FIG. 4.7. Dependence of the quality and cost of the decision tree algorithm on $|S_i|$. The accuracy improves and the cost decreases as $|S_i|$ increases.

experimental results are shown in Figure 4.8. As shown, the effect of the depth is more pronounced on the communication than on the quality of the result. Accuracy increases from 72% to 81% as the depth is increased from 2 to 5. However for a depth of 7, the accuracy of the P^2*DTI* decreases by 2% compared to a depth of 5. The reason for this is overfitting of the domain. As the depth is increased, there is potentially more ties for every majority vote leading to a message explosion. As the depth is increased from 2 to 7, the stationary messages increase from 0.71 to 1.56. The stationary bytes goes up to 58.71, for a depth of 7.

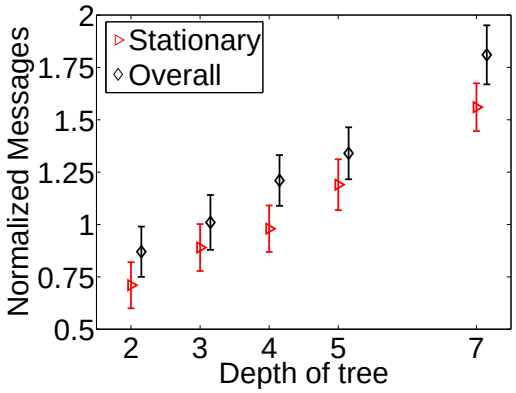
Although the induced tree of depth 5 is around 3% more accurate than the tree of depth 3, we have used trees of depth 3 in all as a baseline. This is because a tree of depth 3 has far lower communication overhead than a tree of depth 5 (0.89 normalized messages for depth of 3 compared to 1.19 normalized messages for depth of 5).

4.5.7 Size of Leaky Bucket

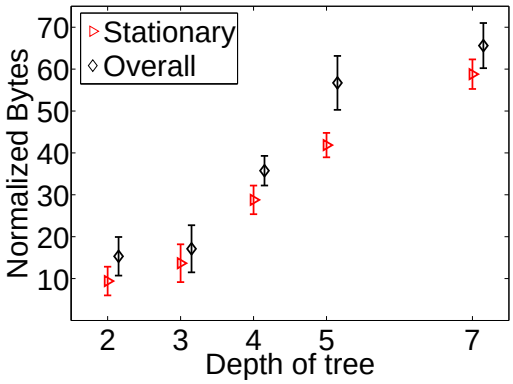
The last algorithm parameter that we have experimented with is the leaky bucket mechanism. In this section we present the effect of the size of the leaky bucket $|L|$ on the accuracy and the cost of the P^2*DTI* algorithm. Figure 4.9 summarizes the effect. As shown in Figure 4.9(a), the stationary accuracy remains constant even as the size of the leaky bucket is made twice or thrice of the edge delay (which is roughly 1100 time units). The overall quality degrades. This is exactly what we expect. As $|L|$ is increased, for every epoch change, the algorithm takes more time to converge, thereby carrying inaccurate results for a longer time. For this reason the overall accuracy degrades. However, once the algorithm adapts to the new distribution, a small number of messages is sufficient to maintain correctness. This is why the leaky bucket has no effect on the stationary accuracy. Contrary to the quality, the cost reduces drastically, from 0.98 stationary messages per peer for $|L|$=500 to 0.71 for $|L|$=4000. Similar is the trend for the bytes transferred.



(a) Quality vs. depth of the tree.

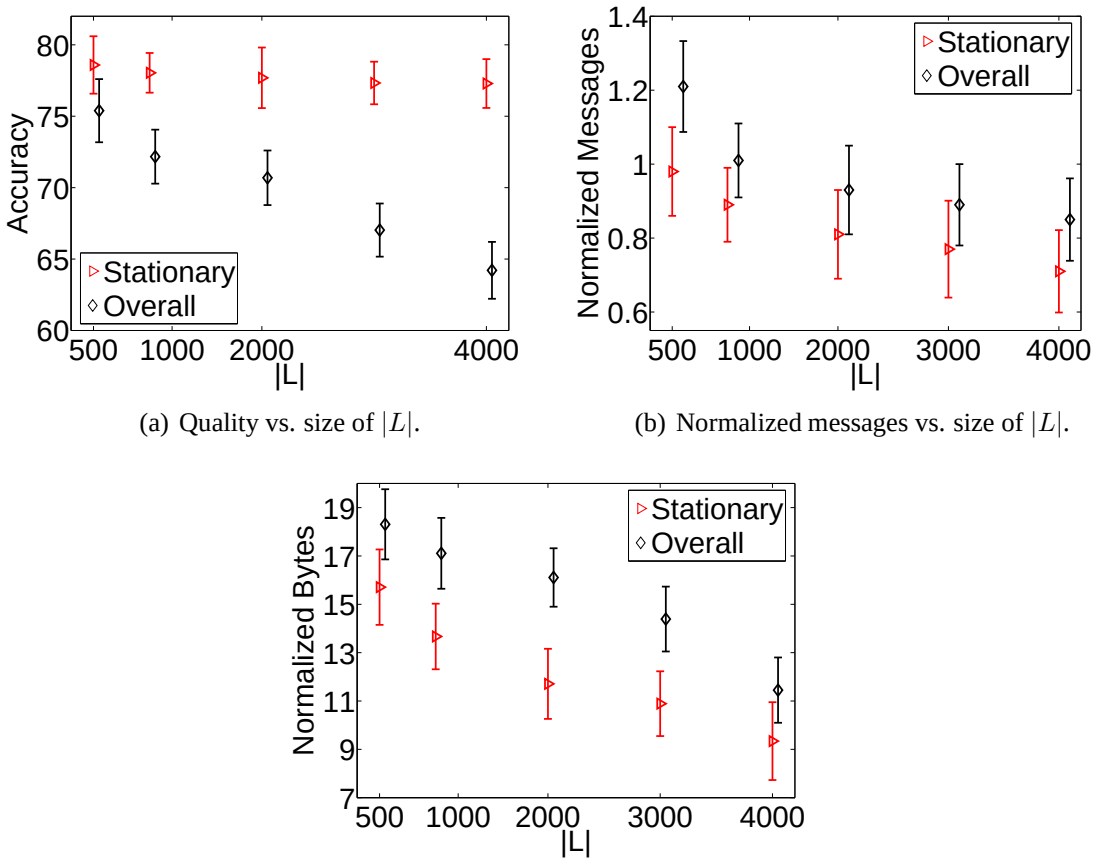


(b) Normalized messages vs. depth of the tree.



(c) Normalized bytes transferred vs. depth of the tree.

FIG. 4.8. Dependence of the quality and cost of the decision tree algorithm on the depth of the induced tree. The accuracy first improves and then degrades as the depth of the tree increases. The cost increases as the depth increases.



(a) Quality vs. size of $|L|$.

(b) Normalized messages vs. size of $|L|$.

(c) Normalized bytes transferred vs. size of $|L|$.

FIG. 4.9. Dependence of the quality and cost of the decision tree algorithm on the size of the leaky bucket.

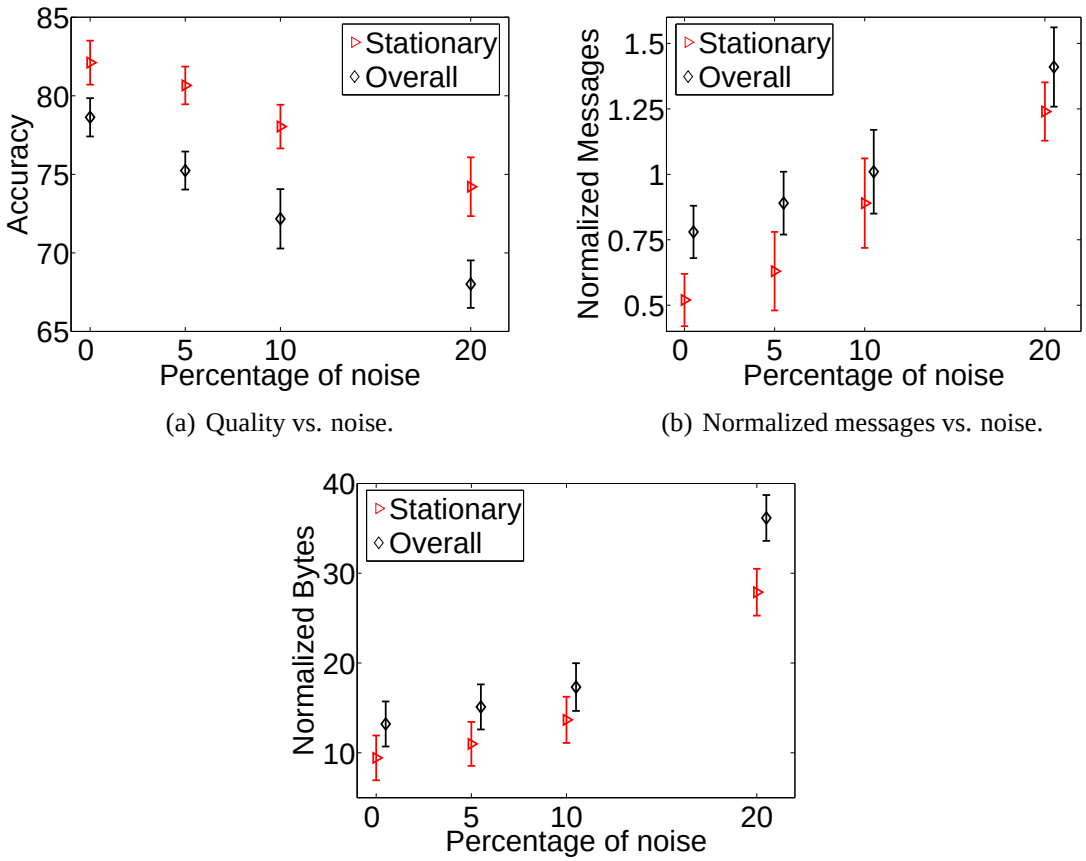
4.5.8 Noise in Data

In this section we vary one of the data parameters — noise. As the noise in the data is increased from 0% to 20%, quality degrades and cost increases. This is demonstrated in Figures 4.10. In Figure 4.10(a), the stationary accuracy decreases from 83% to approximately 75%. The stationary messages increase from 0.52 to 1.24 and stationary bytes increase from 9.43 to 17.89 as demonstrated in Figures 4.10(b) and 4.10(c) respectively. This happens because with increasing noise, every comparison consumes more resources to decide the better one and this decision can often get flipped every time the data changes. As a result, the quality degrades and the cost increases. The important observation here is that even for the highest noise, the number of bytes transferred is 17.89, far less than the maximal allowable rate of 80.

4.5.9 Number of Attributes

The last parameter we varied is the number of attributes. We have measured the effect in three different ways — (1) increasing the number of attributes while keeping the number of tuples constant (Figures 4.11(a), 4.11(b) and 4.11(c)), (2) increasing the number of attributes while increasing the number of tuples linearly with the number of attributes (Figures 4.12(a), 4.12(b) and 4.12(c)), and (3) increasing the number of attributes while increasing the number of tuples linearly with the size of the domain (Figures 4.13(a), 4.13(b) and 4.13(c)).

As the number of the attributes is increased keeping the number of tuples constant (at 500 tuples per peer), the stationary accuracy decreases from 73% to 63% (Figure 4.11(a)). Similarly Figures 4.11(b) and 4.11(c) demonstrate that the normalized messages increase from 0.9 to 1.25 and the normalized bytes increase from approximately 17 to 92. Note that for the number of attributes=10, 15 and 20, the maximal bytes transferred for a broadcast-based algorithm is 80, 120 and 160 respectively.



(a) Quality vs. noise.

(b) Normalized messages vs. noise.

(c) Normalized bytes transferred vs. noise.

FIG. 4.10. Dependence of the quality and cost of the decision tree algorithm on noise in the data. The accuracy decreases and cost increases as percentage of noise increases.

The second set of Figures 4.12(a), 4.12(b) and 4.12(c) show the effect on the number of attributes as the number of tuples is increased linearly with the number of attributes (such that, number of tuples=$50\times$ number of attributes. For 10 attributes we have used 500 data tuples per peer. We have increased it to 750 tuples for 15 attributes and further increased it to 1000 tuples for 20 attributes. The accuracy degrades from 73% to 63%. The more interesting is the effect on the communication. The stationary messages increase very slowly (from 0.89 to 0.92), demonstrating the fact that the algorithm is scalable.

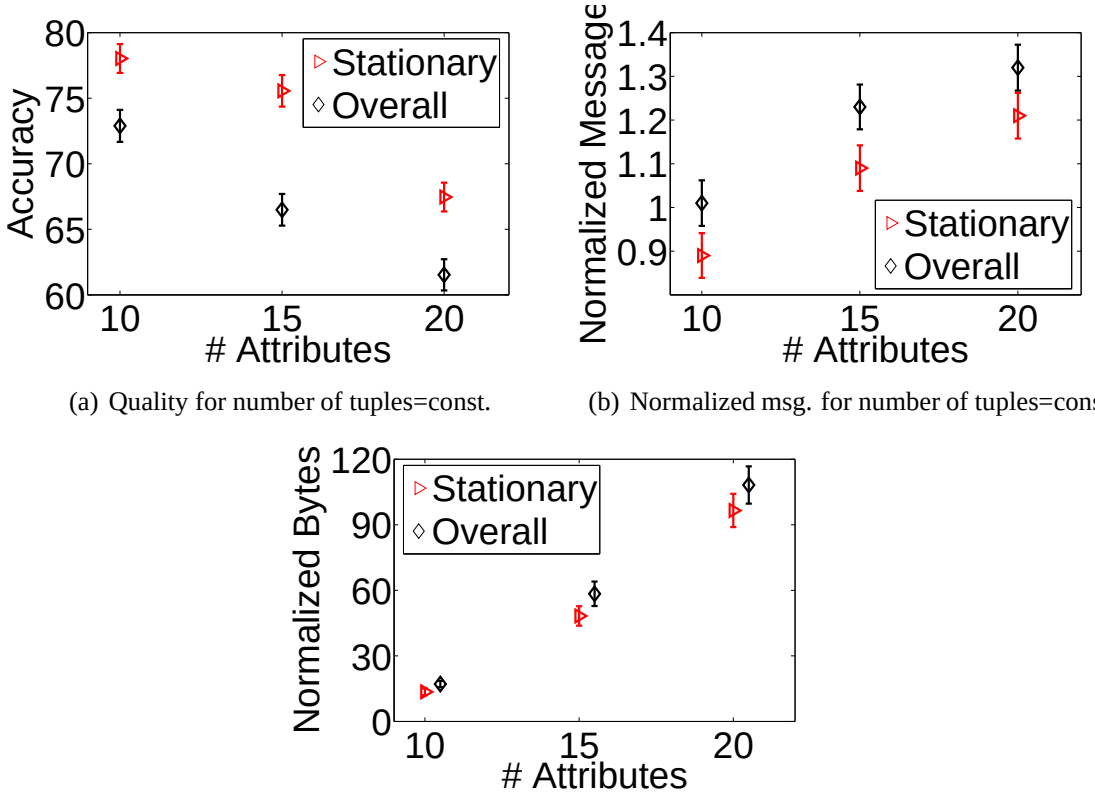
One last variation is the relationship of number of attributes when the number of tuples is increased in proportion to the size of the domain (number of tuples=1%$\times 2^{numberofattributes}$). For 10, 15 and 20 attributes, the number of tuples per peer we used are 10, 330 and 10000 respectively. The accuracy improves and the normalized messages decrease as the number of attributes is increased. The number of bytes transferred increases, though for number of attributes=20, it is still well below what would have been used for a broadcast-based algorithm.

4.5.10 Discussion

In the previous section we have presented the quality and the cost of the P^2*DTI* algorithm on the different algorithm parameters. Our findings can be summarized as follows.

- In most cases P^2*DTI* results in a loss of accuracy over J48. This is because of the simpler gain function that we have chosen. However, due to the heavy communication and synchronization cost of centralizing and applying J48, the observed modest loss of accuracy seems quite reasonable.

- The P^2*DTI* algorithm is highly scalable with respect to the number of peers. As shown by our scalability experiments, increasing the number of peers has little effect on the quality or cost.

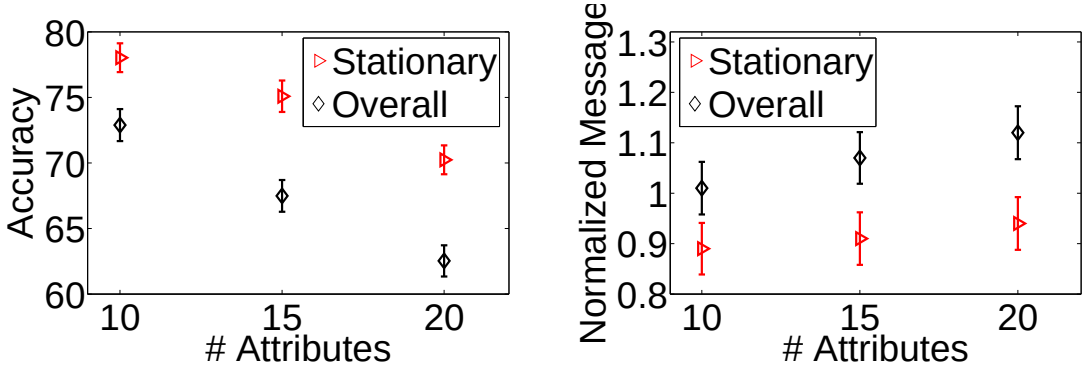


(a) Quality for number of tuples=const.

(b) Normalized msg. for number of tuples=const.

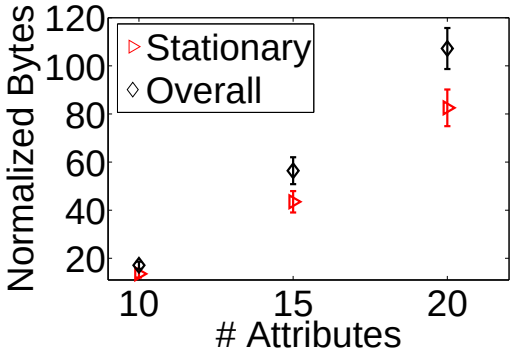
(c) Normalized bytes for number of tuples=const.

FIG. 4.11. Dependence of the quality and cost of the decision tree algorithm on the number of attributes when number of tuples=constant. As the number of attributes increase, the accuracy drops and the cost increases.



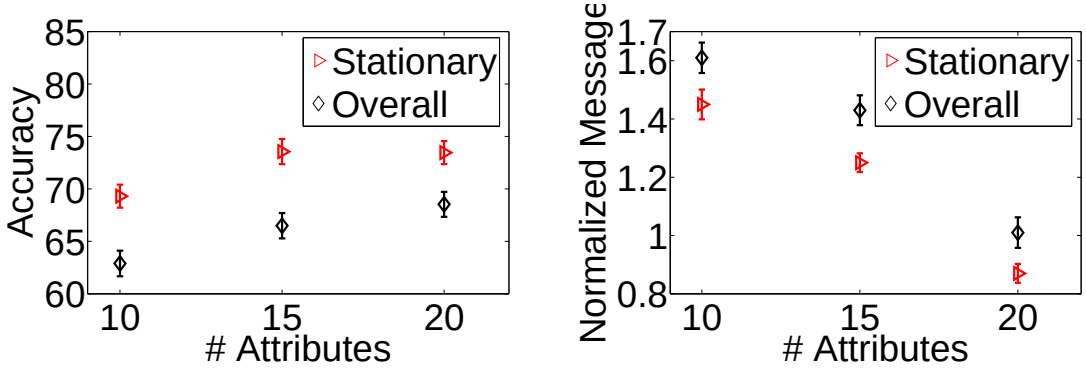
(a) Quality when number of tuples increase linearly with #attr.

(b) Normalized msg. when number of tuples increase linearly with #attr.



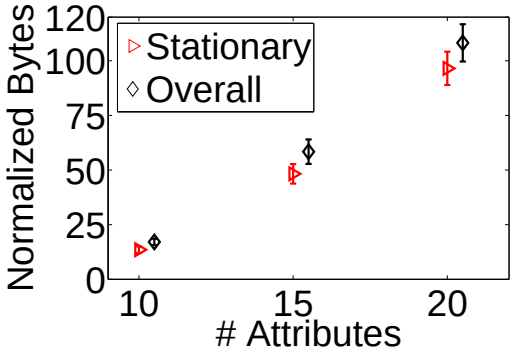
(c) Normalized bytes when number of tuples increase linearly with #attr.

FIG. 4.12. Dependence of the quality and cost of the decision tree algorithm on the number of attributes when #tuples increase linearly with number of attributes.



(a) Quality when number of tuples increase linearly with |domain|.

(b) Normalized msg. number of tuples increase linearly with |domain|.



(c) Normalized bytes number of tuples increase linearly with |domain|.

FIG. 4.13. Dependence of the quality and cost of the decision tree algorithm on the number of attributes when number of tuples increase linearly with |domain|.

- Increasing the number of tuples increases the accuracy of the tree and decreases the cost. Even for tuples per peer = a quarter of the size of the domain (250 tuples per peer for 10 attributes), the monitoring cost is 1.25 — less than the maximal allowable cost of 2.0.

- Note that every increment in the number of attributes doubles the search space. The quality and cost of our algorithm remains moderate even of the number of attributes is doubled.

- Noise in the data degrades the quality and cost — it can be compensated either by increasing the number of tuples or increasing the depth of the tree. Depth of three seems to a be moderate choice since the accuracy is good and the monitoring cost is low as well. Increasing the depth improves the accuracy with a heavy penalty on the cost.

4.6 Summary

In this chapter we presented an asynchronous scalable algorithm for inducing a decision tree in large P2P networks. With sufficient time, the algorithm converges to the same tree given all the data of all the peers. To the best of the authors' knowledge this is one of the first attempts on developing such an algorithm. The algorithm is suitable for scenarios in which the data is distributed across a large P2P network as it seamlessly handles data changes and peer failures. We have conducted extensive experiments with synthetic dataset to analyze the different parameters of the algorithms. The results point out that the algorithm is accurate and suffers moderate communication overhead compared to a broadcast-based algorithm. The algorithm is also highly scalable both with respect to the number of peers and number of attributes.

This chapter relies on the majority voting algorithm as a building block. The majority

voting protocol is a highly efficient and scalable protocol, mainly due to the existence of local pruning rules. In the literature such algorithms are commonly referred to as local algorithms. Previous work on exact local algorithms mainly focused on developing efficient building blocks such as majority voting [196], L2-thresholding [195] and more. In this chapter, similar to [105], we leverage these powerful building blocks to show how more complex data mining algorithms can be developed for large-scale distributed systems. In the process we have also shown how complex functions such as entropy need to be simplified to misclassification gain in order to aid in the algorithm development process.

Chapter 5

CONCLUSIONS AND FUTURE WORK

Proliferation of communication technologies and reduction in storage costs over the past decade have led to the emergence of several distributed systems. Most of the initial effort was directed towards developing systems based on client-server architecture such as most modern web servers, file servers, mail servers, print servers, e-commerce applications and more. These became popular mainly because of the simple synchronous nature of the communication between the server and the client. Since the data is stored mainly at the server, it is relatively easy to ensure security and privacy. However as more and more of these systems evolved, several shortcomings were realized. With increased clients and more requests from each client, traffic congestion between the server and the client became heavy, leading to reduced throughput. Moreover in many cases several clients sat idle, leading to unbalanced load distribution. Another major drawback was the lack of robustness to failed clients. Since in a typical client server-based distributed system the data is not replicated but rather kept at a single site, a node failure is devastating for the entire system.

Peer-to-Peer (P2P) systems were developed to solve some of these problems. P2P systems, as already defined in Chapter 2 Section 2.2, is an architecture where there are no special machine or machines that provide a service or manage the network resources. Instead all responsibilities are uniformly divided among all machines, known as peers. Peers can serve both as clients and servers hence they are often termed as *servants*. While P2P

systems originally emerged as a forum for sharing and exchanging data such as Gnutella, BitTorrents, e-Mule, Kazaa, and Freenet, more recent research such as P2P web community formation argues that the consumers will greatly benefit from the knowledge locked in the data. Algorithms designed for knowledge extraction from typical client-server systems fail to perform well in such massive P2P systems mainly because of (1) asynchronous communication paradigm, (2) decentralized control, (3) large number of nodes and (4) robustness to system failures.

In this dissertation we have proposed several algorithms specifically geared towards P2P systems. More specifically in Chapter 3 our contributions are as follows:

- The *DeFraLC* framework describes a theorem which can be used to build several deterministic local algorithms.
- Based on the above mentioned theorem, we have developed a generic local algorithm (*PeGMA*) capable of computing complex functions defined on the average of the data horizontally distributed over a large P2P network.
- We have also discussed a general framework for monitoring, and consequent reactive updating of any model of horizontally distributed data.
- Finally we describe the application of this framework for the problems of tracking the average of distributed data, the first eigenvector of that data, and the k-means clustering of that data and multivariate regression of the data. Our experimental results have shown that the algorithms are accurate and suffer modest communication cost compared to centralization of the entire data.

Previously, a local algorithm needed to be developed for each data mining problem. The results of Chapter 3 present an easier solution — we can develop several data mining algorithms by following the guidelines presented therein.

In the next chapter, Chapter 4 we have shown how a complex yet well understood data mining algorithm such as decision tree induction can be developed for P2P systems. In the process, we have chosen a relatively simple attribute gain metric such as misclassification gain contrary to more complex ones such as entropy and Gini since (1) it is easier to compute in a P2P domain and (2) achieves comparable accuracy. Also we have shown how the tree can be build in a top down manner without the need for all the peers to synchronize and exchange data.

This thesis has shown how traditional computation problems such as k-means, multivariate regression, decision tree induction can be converted to a decision problem and solved using local algorithms in large distributed environments. We have gained valuable insights which show that it is possible to develop distributed local algorithms for an important class of data mining problems *viz.* decision problems. Therefore several efficient local algorithms can be developed for monitoring the data mining models generated using a centralized computation. It is still unclear to us if local algorithms can be developed for computing the models. In our case we have used an inefficient technique — convergecast-broadcast — to perform the computation and then rely on the efficiency and correctness of the local algorithm. This opens up several interesting research areas. They are listed below.

Complex data mining algorithm development for P2P systems: Previous work on developing deterministic yet efficient algorithms for P2P systems mainly focused on developing basic primitives such as sum computation [119], majority voting [196] etc. In order to harness the power of the next generation P2P systems, complex algorithms need to be developed for extracting knowledge from the data locked in these systems. These primitives might aid in the algorithm development process (as we have shown for our decision tree algorithm in Chapter 4); however in many cases we may need to develop algorithms from scratch. Clustering, classification, association rule mining in P2P systems need efficient and scalable solutions. This work and some other work such as [105] and [116] have

barely scratched the surface compared to the huge number of algorithms that still need to be developed. Furthermore, these algorithms need to be deployed in real life to solve practical problems.

Theoretical analysis of local algorithms for distributed systems: Local algorithms for P2P systems have been an active area of research for the last five years. While a few algorithms have been developed, the field lacks any theoretical work in the "locality" of these algorithms. The local and efficiency claims are, in many cases, intuitive and hence experimental results are the only resource to demonstrate efficiency, locality and scalability. Although in this work we have proposed two definitions of locality, it is far from being perfect. Analysis of the communication complexity, locality and convergence rate for the proposed local algorithms in the literature need to be carried out in order to provide better insights to the working of these algorithms.

Ordinal statistics based algorithm development: In many cases for deterministic local algorithms, tied votes consume the most resources. Consider the example of comparing two real numbers. If the numbers are nearly equal, a lot of resources are consumed by the majority voting algorithm. Ordinal statistics can provide a cheaper solution since, in ordinal decision theory, the comparison is important, not the actual value. Applying order statistics (as done in [37]) to solve no-tied majority vote can be an active area of research.

Local algorithms for computation problems: As already discussed, we have used local algorithms to monitor a model generated using centralized/sampling techniques (except decision tree induction). We still do not have any concrete ideas on how to develop local algorithms for many of the computational problems that we have discussed in this thesis.

Appendix A

MAJORITY VOTING PROTOCOL AND ITS RELATION TO L2 THRESHOLDING ALGORITHM

The majority voting protocol was developed by Wolff and Schuster [196]. In this Appendix we show how the said protocol can be derived from our L2 Thresholding algorithm. The original protocol suffers from one major drawback – small variations of local data of a peer *i.e.* Δ^k triggers more messages. This is detrimental to a dynamic behavior where the data changes frequently. Our mapping of the majority protocol to the L2 algorithm solves this problem as well be changing the rules under which a message is sent and thereby making it more communication efficient under such circumstances. The original majority protocol is presented in Algorithm 13.

Recalling the notations we have used in Chapter 3, $\overrightarrow{\mathcal{K}_k}$, $\overrightarrow{\mathcal{A}_{k,\ell}}$ and $\overrightarrow{\mathcal{W}_{k,\ell}}$ refer to the average knowledge, agreement and witheld knowledge of peer P_k,

Now for any majority threshold λ,

1. $\Delta^k = \left(\overrightarrow{\mathcal{K}_k} - \lambda\right)|\mathcal{K}_k| \Leftrightarrow \overrightarrow{\mathcal{K}_k} = \lambda + \frac{\Delta^k}{|\mathcal{K}_k|}$
2. $\Delta^{k\ell} = \left(\overrightarrow{\mathcal{A}_{k,\ell}} - \lambda\right)|\mathcal{A}_{k,\ell}| \Leftrightarrow \overrightarrow{\mathcal{A}_{k,\ell}} = \lambda + \frac{\Delta^{k\ell}}{|\mathcal{A}_{k,\ell}|}$
3. $\overrightarrow{\mathcal{W}_{k,\ell}} = \frac{\overrightarrow{\mathcal{K}_k}|\mathcal{K}_k| - \overrightarrow{\mathcal{A}_{k,\ell}}|\mathcal{A}_{k,\ell}|}{|\mathcal{K}_k| - |\mathcal{A}_{k,\ell}|} = \frac{\Delta^k + \lambda|\mathcal{K}_k| - \Delta^{k\ell} - \lambda|\mathcal{A}_{k,\ell}|}{|\mathcal{K}_k| - |\mathcal{A}_{k,\ell}|} = \frac{\Delta^k - \Delta^{k\ell}}{|\mathcal{K}_k| - |\mathcal{A}_{k,\ell}|} + \lambda$

The condition for sending messages in majority voting can be written as:

Input: δ^k, N^k, L
Output: if $\Delta^k \geq 0$ then 1 else 0
Local variables: $\forall \ell \in N^k : \delta^{\ell k}, \delta^{k\ell}$
Definitions: $\Delta^k = \delta^k + \sum_{\ell \in N^k} \delta^{\ell k}$, $\Delta^{k\ell} = \delta^{k\ell} + \delta^{\ell k}$
Initialization:
begin
    forall $\ell \in N^k$ **do**
        $\delta^{k\ell} = \delta^{\ell k} = 0$;
        SendMessage(ℓ);
    end
end
if MessageRecvd$\left(P^\ell, \delta\right)$ **then** $\delta^{\ell k} \leftarrow \delta$;
if NodeFailure$\left(\ell \in N^k\right)$ **then** $N^k \leftarrow N^k \setminus \{\ell\}$;
if AddNeighbor$\left(\ell \in N^k\right)$ **then** $N^k \leftarrow N^k \cup \{\ell\}$;
if N^k, δ^k *changes or MessageRecvd* **then** call **OnChange**();
Function ***OnChange()***
begin
    forall $\ell \in N^k$ **do**
        if $\left(\Delta^{k\ell} \geq 0 \wedge \Delta^{k\ell} > \Delta^k\right) \vee \left(\Delta^{k\ell} < 0 \wedge \Delta^{k\ell} < \Delta^k\right)$ **then**
            call **SendMessage**(ℓ);
        end
    end
end
Function ***SendMessage(ℓ)***
begin
    if $time\,() - last\_message \geq L$ **then**
        $\delta^{k\ell} \leftarrow (\Delta^k - \delta^{\ell k})$;
        $last\_message \leftarrow time\,()$;
        Send $\left\langle \delta^{k\ell} \right\rangle$ to ℓ;
    end
    else Wait $L - (time\,() - last\_message)$ time units and then call **OnChange**();
end

Algorithm 13: Local Majority Vote

$$\left[\lambda \geq \Delta^{k,\ell} \geq \Delta^{k}\right] \vee \left[\lambda < \Delta^{k,\ell} \leq \Delta^{k}\right]$$

Similarly, the condition for sending messages in L2 thresholding can be written as:

$$\left[\lambda \geq \overrightarrow{\mathcal{A}_{k,\ell}} \wedge \lambda \geq \overrightarrow{\mathcal{W}_{k,\ell}}\right] \vee \left[\lambda < \overrightarrow{\mathcal{A}_{k,\ell}} \wedge \lambda < \overrightarrow{\mathcal{W}_{k,\ell}}\right]$$

Consider the first condition.

$$\begin{aligned}
& \left[\lambda \geq \overrightarrow{\mathcal{A}_{k,\ell}} \wedge \lambda \geq \overrightarrow{\mathcal{W}_{k,\ell}}\right] \\
\Rightarrow \quad & \left[\lambda \geq \frac{\Delta^{k,\ell}}{|\mathcal{A}_{k,\ell}|} + \lambda \wedge \lambda \geq \frac{\Delta^{k} - \Delta^{k,\ell}}{|\mathcal{K}_k| - |\mathcal{A}_{k,\ell}|} + \lambda\right] \\
\Rightarrow \quad & \left[0 \geq \frac{\Delta^{k,\ell}}{|\mathcal{A}_{k,\ell}|} \wedge 0 \geq \frac{\Delta^{k} - \Delta^{k,\ell}}{|\mathcal{K}_k| - |\mathcal{A}_{k,\ell}|}\right] \\
\Rightarrow \quad & \left[0 \geq \Delta^{k,\ell} \wedge 0 \geq \Delta^{k} - \Delta^{k,\ell}\right] \\
\Rightarrow \quad & \left[0 \geq \Delta^{k,\ell} \geq \Delta^{k}\right]
\end{aligned}$$

The last equation shows that the condition for sending messages in the majority protocol is a special case of the same in the L2 algorithm. Consequently, we can use the same rule that we used in L2 in determining how much of data to send. In L2 instead of setting $\overrightarrow{\mathcal{A}_{k,\ell}}$ to $\overrightarrow{\mathcal{K}_k}$, we set it to a point between the current direction of $\overrightarrow{\mathcal{A}_{k,\ell}}$ and $\overrightarrow{\mathcal{K}_k}$. Similarly, in majority voting we should set $\Delta^{k,\ell} = \alpha\Delta^{k}$. This way, subsequent small variations in Δ^{k} will not trigger any more messages. The modified pseudo-code is presented in Algorithm 14.

Input: δ^k, N^k, L, α
Output: if $\Delta^k \geq 0$ then 1 else 0
Local variables: $\forall \ell \in N^k : \delta^{\ell k}, \delta^{k\ell}$
Definitions: $\Delta^k = \delta^k + \sum_{\ell \in N^k} \delta^{\ell k}$, $\Delta^{k\ell} = \delta^{k\ell} + \delta^{\ell k}$
Initialization:
begin
    forall $\ell \in N^k$ **do**
        $\delta^{k\ell} = \delta^{\ell k} = 0$;
        SendMessage(ℓ);
    end
end
if MessageRecvd$\big(P^\ell, \delta\big)$ **then** $\delta^{\ell k} \leftarrow \delta$;
if NodeFailure$\big(\ell \in N^k\big)$ **then** $N^k \leftarrow N^k \setminus \{\ell\}$;
if AddNeighbor$\big(\ell \in N^k\big)$ **then** $N^k \leftarrow N^k \cup \{\ell\}$;
if N^k, δ^k *changes or MessageRecvd* **then** call **OnChange**();
Function ***OnChange()***
begin
    forall $\ell \in N^k$ **do**
        if $\big(\Delta^{k\ell} \geq 0 \wedge \Delta^{k\ell} > \Delta^k\big) \vee \big(\Delta^{k\ell} < 0 \wedge \Delta^{k\ell} < \Delta^k\big)$ **then**
            call **SendMessage**(ℓ);
        end
    end
end
Function ***SendMessage(ℓ)***
begin
    if $time\,() - last\_message \geq L$ **then**
        $\delta^{k\ell} \leftarrow (\alpha\Delta^k - \delta^{\ell k})$;
        $last\_message \leftarrow time\,()$;
        Send $\big\langle \delta^{k\ell} \big\rangle$ to ℓ;
    end
    else Wait $L - (time\,() - last\_message)$ time units and then call **OnChange**();
end

Algorithm 14: Dynamic Local Majority Vote

Appendix B

INFORMATION GAIN COMPARISON

In this appendix first we show how the misclassification gain difference between any two arbitrary attributes A^i and A^j and binary class label $C = 0$ and $C = 1$ can be posed as the difference of sums of the entries of their cross tables. Then we derive the formula for zero thresholding the gini information gain for two binary attributes.

B.1 Notation

We follow the same notations as in Section 4.4.1. Let S be a set of learning examples — each a vector in $\{0, 1\}^d \times \{0, 1\}$ — where the first d entries of each example denote the attributes, $A^1, \ldots, A^d$, and the additional one denotes the class C. Here we are interested in comparing arbitrarily two attributes A^i and A^j. Let x^i_{k0} denote the number of examples in the set S for which $A^i = k$ and $C = 0\ \forall k \in [0 \ldots (m^i - 1)]$. The following table (Table B.1) shows all the possible values of the attributes and the class.

Similarly for A^j the values are shown in Table B.2.

B.2 Thresholding Misclassification Gain

In this section we show that the misclassification gain between two attributes A^i and A^j can be written as the difference of sums of the entries in their cross tables. The following theorem states it more formally.

Attribute value (A^i)	C=0	C=1	(C=0)+(C=1)
0	x^i_{00}	x^i_{01}	$x^i_{0\cdot}$
1	x^i_{10}	x^i_{11}	$x^i_{1\cdot}$
$\vdots$	$\vdots$	$\vdots$	$\vdots$
m^i-1	$x^i_{(m^i-1)0}$	$x^i_{(m^i-1)1}$	$x^i_{(m^i-1)\cdot}$

Table B.1. Number of entries of attribute A^i and the class C.

Attribute value (A^j)	C=0	C=1	(C=0)+(C=1)
0	x^j_{00}	x^j_{01}	$x^j_{0\cdot}$
1	x^j_{10}	x^j_{11}	$x^j_{1\cdot}$
$\vdots$	$\vdots$	$\vdots$	$\vdots$
m^j-1	$x^j_{(m^j-1)0}$	$x^j_{(m^j-1)1}$	$x^j_{(m^j-1)\cdot}$

Table B.2. Number of entries of attribute A^j and the class C.

Theorem B.2.1. *Given two attributes A^i and A^j, thresholding the misclassification gain difference against zero is given by $\sum_{k=0}^{m^i-1}|x^i_{k0}-x^i_{k1}|-\sum_{\ell=0}^{m^j-1}\left|x^j_{\ell 0}-x^j_{\ell 1}\right|>0$.*

Proof. As given in the table, let $x^i_{k\cdot}$ denote the number of examples in which $A^i=k$ both for $C=0$ and $C=1$ i.e. $x^i_{k\cdot}=x^i_{k0}+x^i_{k1}$. Similarly, let $x^j_{\ell\cdot}$ denotes the same for A^j. Note that, $\sum_{k=0}^{m^i-1}x^i_{k\cdot}=\sum_{\ell=0}^{m^j-1}x^j_{\ell\cdot}=|S|$. The attribute gain difference between A^i and A^j is given as:

$$
\begin{aligned}
&= \sum_{k=0}^{m^i-1} \frac{x^i_{k\cdot}}{|S|}\left[\frac{1}{2} - \left|\frac{x^i_{k0}}{x^i_{k\cdot}} - \frac{1}{2}\right|\right] - \sum_{\ell=0}^{m^j-1} \frac{x^j_{\ell\cdot}}{|S|}\left[\frac{1}{2} - \left|\frac{x^j_{\ell 0}}{x^j_{\ell\cdot}} - \frac{1}{2}\right|\right] \\
&= \sum_{k=0}^{m^i-1} \frac{x^i_{k\cdot}}{|S|}\left[\frac{1}{2} - \left|\frac{2x^i_{k0} - x^i_{k\cdot}}{2x^i_{k\cdot}}\right|\right] - \sum_{\ell=0}^{m^j-1} \frac{x^j_{\ell\cdot}}{|S|}\left[\frac{1}{2} - \left|\frac{2x^j_{\ell 0} - x^j_{\ell\cdot}}{2x^j_{\ell\cdot}}\right|\right] \\
&= \left[\sum_{k=0}^{m^i-1} \frac{x^i_{k\cdot}}{2|S|} - \sum_{\ell=0}^{m^j-1} \frac{x^j_{\ell\cdot}}{2|S|}\right] + \sum_{\ell=0}^{m^j-1} \frac{x^j_{\ell\cdot}}{|S|}\left|\frac{2x^j_{\ell 0} - x^j_{\ell\cdot}}{2x^j_{\ell\cdot}}\right| - \sum_{k=0}^{m^i-1} \frac{x^i_{k\cdot}}{|S|}\left|\frac{2x^i_{k0} - x^i_{k\cdot}}{2x^i_{k\cdot}}\right| \\
&= \left[\frac{|S|}{2|S|} - \frac{|S|}{2|S|}\right] + \frac{1}{2|S|}\left[\sum_{\ell=0}^{m^j-1} \left|2x^j_{\ell 0} - x^j_{\ell\cdot}\right| - \sum_{k=0}^{m^i-1} \left|2x^i_{k0} - x^i_{k\cdot}\right|\right] \\
&= \frac{1}{2|S|}\left[\sum_{\ell=0}^{m^j-1} \left|x^j_{\ell 0} - x^j_{\ell 1}\right| - \sum_{k=0}^{m^i-1} \left|x^i_{k0} - x^i_{k1}\right|\right]
\end{aligned}
$$

The last equality follows from the fact that $x^j_{\ell\cdot} = x^j_{\ell 0} + x^j_{\ell 1}$ and $x^i_{k\cdot} = x^i_{k0} + x^i_{k1}$. Therefore, zero thresholding the attribute gain difference is the same as zero thresholding the above quantity:

$$\textstyle\sum_{k=0}^{m^i-1} \left|x^i_{k0} - x^i_{k1}\right| - \sum_{\ell=0}^{m^j-1} \left|x^j_{\ell 0} - x^j_{\ell 1}\right| > 0$$

□

B.3 Thresholding Gini Information Gain

In this section we show that thresholding the Gini information gain is more complicated than thresholding the misclassification error.

Theorem B.3.1. *Let A^i and A^j be two binary attributes. Thresholding the Gini information gain difference of A^i and A^j against zero is the same as checking if $\left[x^j_{01}\right]^2 x^j_{1\cdot} x^i_{0\cdot} x^i_{1\cdot} + \left[x^j_{11}\right]^2 x^j_{0\cdot} x^i_{0\cdot} x^i_{1\cdot} - \left[x^i_{01}\right]^2 x^i_{1\cdot} x^j_{0\cdot} x^j_{1\cdot} - \left[x^i_{11}\right]^2 x^i_{0\cdot} x^j_{0\cdot} x^j_{1\cdot} > 0$.*

Proof. The Gini information gain difference of A^i and A^j is:

$$
\begin{aligned}
&= x^i_{0\cdot}\left[1-\left(\frac{x^i_{00}}{x^i_{0\cdot}}\right)^2-\left(\frac{x^i_{01}}{x^i_{0\cdot}}\right)^2\right]+x^i_{1\cdot}\left[1-\left(\frac{x^i_{10}}{x^i_{1\cdot}}\right)^2-\left(\frac{x^i_{11}}{x^i_{1\cdot}}\right)^2\right]\\
&\quad -x^j_{0\cdot}\left[1-\left(\frac{x^j_{00}}{x^j_{0\cdot}}\right)^2-\left(\frac{x^j_{01}}{x^j_{0\cdot}}\right)^2\right]-x^j_{1\cdot}\left[1-\left(\frac{x^j_{10}}{x^j_{1\cdot}}\right)^2-\left(\frac{x^j_{11}}{x^j_{1\cdot}}\right)^2\right]\\
&= \left[x^i_{0\cdot}-\frac{\left(x^i_{00}\right)^2}{x^i_{0\cdot}}-\frac{\left(x^i_{01}\right)^2}{x^i_{0\cdot}}\right]+\left[x^i_{1\cdot}-\frac{\left(x^i_{10}\right)^2}{x^i_{1\cdot}}-\frac{\left(x^i_{11}\right)^2}{x^i_{1\cdot}}\right]\\
&\quad -\left[x^j_{0\cdot}-\frac{\left(x^j_{00}\right)^2}{x^j_{0\cdot}}-\frac{\left(x^j_{01}\right)^2}{x^j_{0\cdot}}\right]-\left[x^j_{1\cdot}-\frac{\left(x^j_{10}\right)^2}{x^j_{1\cdot}}-\frac{\left(x^j_{11}\right)^2}{x^j_{1\cdot}}\right]\\
&= x^i_{0\cdot}+x^i_{1\cdot}-x^j_{0\cdot}-x^j_{1\cdot}\\
&\quad +\frac{\left(x^j_{00}\right)^2+\left(x^j_{01}\right)^2}{x^j_{0\cdot}}+\frac{\left(x^j_{10}\right)^2+\left(x^j_{11}\right)^2}{x^j_{1\cdot}}-\frac{\left(x^i_{00}\right)^2+\left(x^i_{01}\right)^2}{x^i_{0\cdot}}-\frac{\left(x^i_{10}\right)^2+\left(x^i_{11}\right)^2}{x^i_{1\cdot}}\\
&= |S|-|S|+\frac{\left[x^j_{00}+x^j_{01}\right]^2-2x^j_{00}x^j_{01}}{x^j_{0\cdot}}+\frac{\left[x^j_{10}+x^j_{11}\right]^2-2x^j_{10}x^j_{11}}{x^j_{1\cdot}}\\
&\quad -\frac{\left[x^i_{00}+x^i_{01}\right]^2-2x^i_{00}x^i_{01}}{x^i_{0\cdot}}-\frac{\left[x^i_{10}+x^i_{11}\right]^2-2x^i_{10}x^i_{11}}{x^i_{1\cdot}}\\
&= \frac{\left[x^j_{0\cdot}\right]^2-2x^j_{00}x^j_{01}}{x^j_{0\cdot}}+\frac{\left[x^j_{1\cdot}\right]^2-2x^j_{10}x^j_{11}}{x^j_{1\cdot}}-\frac{\left[x^i_{0\cdot}\right]^2-2x^i_{00}x^i_{01}}{x^i_{0\cdot}}-\frac{\left[x^i_{1\cdot}\right]^2-2x^i_{10}x^i_{11}}{x^i_{1\cdot}}\\
&= \frac{\left[x^j_{0\cdot}\right]^2-2\left[x^j_{0\cdot}-x^j_{01}\right]x^j_{01}}{x^j_{0\cdot}}+\frac{\left[x^j_{1\cdot}\right]^2-2\left[x^j_{1\cdot}-x^j_{11}\right]x^j_{11}}{x^j_{1\cdot}}\\
&\quad -\frac{\left[x^i_{0\cdot}\right]^2-2\left[x^i_{0\cdot}-x^i_{01}\right]x^i_{01}}{x^i_{0\cdot}}-\frac{\left[x^i_{1\cdot}\right]^2-2\left[x^i_{1\cdot}-x^i_{11}\right]x^i_{11}}{x^i_{1\cdot}}\\
&= x^j_{0\cdot}-2x^j_{01}+\frac{2\left[x^j_{01}\right]^2}{x^j_{0\cdot}}+x^j_{1\cdot}-2x^j_{11}+\frac{2\left[x^j_{11}\right]^2}{x^j_{1\cdot}}\\
&\quad -x^i_{0\cdot}+2x^i_{01}-\frac{2\left[x^i_{01}\right]^2}{x^i_{0\cdot}}-x^i_{1\cdot}+2x^i_{11}-\frac{2\left[x^i_{11}\right]^2}{x^i_{1\cdot}}\\
&= \left[x^j_{0\cdot}+x^j_{1\cdot}-x^i_{0\cdot}-x^i_{1\cdot}\right]+2\left[x^i_{01}+x^i_{11}-x^j_{01}-x^j_{11}\right]\\
&+\ 2\left[\frac{\left[x^j_{01}\right]^2}{x^j_{0\cdot}}+\frac{\left[x^j_{11}\right]^2}{x^j_{1\cdot}}-\frac{\left[x^i_{01}\right]^2}{x^i_{0\cdot}}-\frac{\left[x^i_{11}\right]^2}{x^i_{1\cdot}}\right]\\
&= \left[|S|-|S|\right]+2\left[(\#class=1)-(\#class=1)\right]+2\left[\frac{\left[x^j_{01}\right]^2}{x^j_{0\cdot}}+\frac{\left[x^j_{11}\right]^2}{x^j_{1\cdot}}-\frac{\left[x^i_{01}\right]^2}{x^i_{0\cdot}}-\frac{\left[x^i_{11}\right]^2}{x^i_{1\cdot}}\right]\\
&= 2\left[\frac{\left[x^j_{01}\right]^2}{x^j_{0\cdot}}+\frac{\left[x^j_{11}\right]^2}{x^j_{1\cdot}}-\frac{\left[x^i_{01}\right]^2}{x^i_{0\cdot}}-\frac{\left[x^i_{11}\right]^2}{x^i_{1\cdot}}\right]
\end{aligned}
$$

Therefore, thresholding the gini information gain difference about zero is the same as determining if the following expression is greater than zero

$$\frac{\left[x_{01}^{j}\right]^{2}}{x_{0.}^{j}}+\frac{\left[x_{11}^{j}\right]^{2}}{x_{1.}^{j}}-\frac{\left[x_{01}^{i}\right]^{2}}{x_{0.}^{i}}-\frac{\left[x_{11}^{i}\right]^{2}}{x_{1.}^{i}}>0$$

which is equalvalent to determining if the following polynomial is greater than zero

$$\left[x_{01}^{j}\right]^{2}x_{1.}^{j}x_{0.}^{i}x_{1.}^{i}+\left[x_{11}^{j}\right]^{2}x_{0.}^{j}x_{0.}^{i}x_{1.}^{i}-\left[x_{01}^{i}\right]^{2}x_{1.}^{i}x_{0.}^{j}x_{1.}^{j}-\left[x_{11}^{i}\right]^{2}x_{0.}^{i}x_{0.}^{j}x_{1.}^{j}>0$$

□

REFERENCES

[1] Y. Afek, S. Kutten, and M. Yung. The Local Detection Paradigm and Its Application to Self-Stabilization. *In Theoretical Computer Science*, 186(1–2):199–229, October 1997.

[2] Gul Agha. *Actors: A Model of Concurrent Computation in Distributed Systems*. MIT Press, Cambridge, MA, USA, 1986.

[3] Amazon Music Recommendation. `http://www.amazon.com/`.

[4] S. Bailey, R. Grossman, H. Sivakumar, and A. Turinsky. Papyrus: a System for Data Mining Over Local and Wide Area Clusters and Super-Clusters. In *Proceedings of CDROM'99*, page 63, New York, NY, USA, 1999.

[5] T. Balch and R. C. Arkin. Motor Schema-based Formation Control for Multiagent Robot Teams. In *Proceedings of ICMAS'95*, pages 10–16, San Francisco, CA, USA, 1995.

[6] Wolf-Tilo Balke, Wolfgang Nejdl, Wolf Siberski, and Uwe Thaden. Progressive Distributed Top-k Retrieval in Peer-to-Peer Networks. In *Proceedings of ICDE'05*, pages 174–185, Tokyo, Japan, 2005.

[7] Sanghamitra Bandyopadhyay, Chris Giannella, Ujjwal Maulik, Hillol Kargupta, Kun Liu, and Souptik Datta. Clustering Distributed Data Streams in Peer-to-Peer Environments. *Information Science*, 176(14):1952–1985, 2006.

[8] A. Bar-Or, D. Keren, A. Schuster, and R. Wolff. Hierarchical Decision Tree Induction in Distributed Genomic Databases. *IEEE Transactions on Knowledge and Data*

Engineering – Special Issue on Mining Biological Data, 17(8):1138 – 1151, August 2005.

[9] A. Bar-Or, D. Keren, A. Schuster, and R. Wolff. Hierarchical Decision Tree Induction in Distributed Genomic Databases. *IEEE Transactions on Knowledge and Data Engineering – Special Issue on Mining Biological Data*, 17(8):1138 – 1151, August 2005.

[10] Valmir C. Barbosa. *An Introduction to Distributed Algorithms*. The MIT Press, 1996.

[11] Mayank Bawa, Aristides Gionis, Hector Garcia-Molina, and Rajeev Motwani. The Price of Validity in Dynamic Networks. In *Proceedings of SIGMOD'04*, pages 515–526, Paris, France, June 2004.

[12] Mikhail Bessonov, Udo Heuser, Igor Nekrestyanov, and Ahmed Patel. Open Architecture for Distributed Search Systems. *Lecture Notes in Computer Science*, 1597:55–69, 1999.

[13] K. Bhaduri and H. Kargupta. An efficient local Algorithm for Distributed Multivariate Regression in Peer-to-Peer Networks. In *Accepted for publication at SDM'08*, 2008.

[14] Y. Birk, L. Liss, A. Schuster, and R. Wolff. A Local Algorithm for Ad Hoc Majority Voting via Charge Fusion. In *Proceedings of ICDCS'04*, 2004.

[15] Yitzhak Birk, Idit Keidar, Liran Liss, Assaf Schuster, and Ran Wolff. Veracity Radius - Capturing the Locality of Distributed Computations. In *Proceedings of PODC'06*, pages 102–111, 2006.

[16] BitTorrents homepage. `http://www.bittorrent.com`.

[17] G. Bontempi and Y.-A Borgne. An Adaptive Modular Approach to the Mining of Sensor Network Data. In *Proceedings of 1st International Workshop on Data Mining in Sensor Networks*, pages 3–9, 2005.

[18] S. Boyd, A. Ghosh, B. Prabhakar, and D. Shah. Gossip Algorithms: Design, Analysis and Applications. In *Proceddings Infocom'05*, pages 1653–1664, Miami, March 2005.

[19] J. Branch, B. Szymanski, C. Gionnella, R. Wolff, and H. Kargupta. In-Network Outlier Detection in Wireless Sensor Networks. In *Proceedings of ICDCS'06*, Lisbon, Portugal, July 2006.

[20] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. *Classification and Regression Trees*. Wadsworth, Belmont, 1984.

[21] Leo Breiman. Bagging Predictors. *Machine Learning*, 24(2):123–140, 1996.

[22] Leo Breiman. Random Forests. *Machine Learning*, 45(1):5–32, 2001.

[23] BRITE. `http://www.cs.bu.edu/brite/`.

[24] Lawrence Bull, Terence C. Fogarty, and M. Snaith. Evolution in Multi-agent Systems: Evolving Communicating Classifier Systems for Gait in a Quadrupedal Robot. In *Proceedings of ICGA'95*, pages 382–388, San Francisco, CA, USA, 1995.

[25] J. Callan. Distributed Information Retrieval. In W.B. Croft, editor, *Advances in Information Retrieval*, chapter 5, pages 127–150. Kluwer Academic Publishers, 2000.

[26] Mario Cannataro, Antonio Congiusta, Andrea Pugliese, Domenico Talia, and Paolo Trunfio. Distributed Data Mining on Grids: Services, Tools, and Applications. *Ieee Transactions On Systems, Man, And CyberneticsPart B: Cybernetics*, 34(6):2465–2451, 2004.

[27] Mario Cannataro and Domenico Talia. The Knowledge Grid. *Communication of ACM*, 46(1):89–93, 2003.

[28] Pei Cao and Zhe Wang. Efficient top-K Query Calculation in Distributed Networks. In *Proceedings of PODC'04*, pages 206–215, St. John's, Newfoundland, Canada, 2004.

[29] D. Caragea, A. Silvescu, and V. Honavar. A Framework for Learning from Distributed Data Using Sufficient Statistics and Its Application to Learning Decision Trees. *International Journal of Hybrid Intelligent Systems*, 1(1-2):80–89, 2004.

[30] Philip K. Chan and Salvatore J. Stolfo. Experiments on Multistrategy Learning by Meta-Learning. In *Proceedings of CIKM'93*, pages 314–323, 1993.

[31] Phillip K. Chan and Salvatore J. Stolfo. Toward Parallel and Distributed Learning by Meta-Learning. In *Working Notes of AAAI Workshop on Knowledge Discovery in Databases*, pages 227–240, 1993.

[32] J. Chen, D. DeWitt, F. Tian, and Y. Wang. NiagaraCQ: a Scalable Continuous Query System for Internet Databases. In *Proceedings of SIGMOD'00*, pages 379–390, Dallas, Texas, 2000.

[33] R. Chen, K. Sivakumar, and H. Kargupta. Collective Mining of Bayesian Networks from Distributed Heterogeneous Data. *Knowledge and Information Systems*, 6(2):164–187, 2004.

[34] W. Chen, J. C. Hou, and L. Sha. Dynamic Clustering for Acoustic Target Tracking in Wireless Sensor Networks. *IEEE Transactions on Mobile Computing*, 3(3):258–271, 2004.

[35] Chinook Bioinformatic homepage. `http://smweb.bcgsc.bc.ca/chinook/`.

[36] Francisco Matias Cuenca-Acuna and Thu D. Nguyen. Text-Based Content Search and Retrieval in Ad-hoc P2P Communities. In *Revised Papers from the NETWORKING 2002 Workshops on Web Engineering and Peer-to-Peer Computing*, pages 220–234, London, UK, 2002.

[37] Kamalika Das, Kanishka Bhaduri, Kun Liu, and Hillol Kargupta. Distributed Identification of Top-l Inner Product Elements and its Application in a Peer-to-Peer Network. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 20(4):475–488, 2008.

[38] The DataGrid Project. `http://eu-datagrid.web.cern.ch/eu-datagrid/default.htm`.

[39] S. Datta, K. Bhaduri, C. Giannella, R. Wolff, and H. Kargupta. Distributed Data Mining in Peer-to-Peer Networks. *IEEE Internet Computing Special Issue on Distributed Data Mining*, 10(4):18–26, 2006.

[40] S. Datta, C. Giannella, and H. Kargupta. K-Means Clustering over Large, Dynamic Networks. In *Proceedings of SDM'06*, pages 153–164, Maryland, 2006.

[41] Souptik Datta and Hillol Kargupta. Uniform Data Sampling from a Peer-to-Peer Network. In *Proceedings of ICDCS'02*, page 50, Toronto, Ontario, 2007.

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

[43] Distributed Data Mining Bibliography homepage at UMBC. `http://www.cs.umbc.edu/~hillol/DDMBIB/`.

[44] DDMT. `http://www.umbc.edu/ddm/wiki/software/DDMT/`.

[45] K. S. Decker. Distributed Problem Solving Techniques: A survey. *IEEE Transactions on Systems Man Cybernetics*, 17(5):729–740, 1987.

[46] Inderjit S. Dhillon and Dharmendra S. Modha. A Data-Clustering Algorithm on Distributed Memory Multiprocessors. In *Revised Papers from Large-Scale Parallel Data Mining, Workshop on Large-Scale Parallel KDD Systems, SIGKDD*, pages 245–260, London, UK, 2000.

[47] Distribustream Peercasting Service homepage. `http://distribustream.org/`.

[48] Pedro Domingos and Geoff Hulten. Mining High-Speed Data Streams. In *Proceedings of KDD'00*, pages 71–80, Boston, MA, 2000.

[49] E. H. Durfee. Blissful ignorance: Knowing just enough to coordinate well. In *Proceedings of ICMAS'95*, pages 406–413, 1996.

[50] E. H. Durfee, V. R. Lesser, and D. D. Corkill. Coherent Cooperation Among Communicating Problem Solvers. *IEEE Transactions on Computers*, 36(11):1275–1291, 1987.

[51] E. H. Durfee, V. R. Lesser, and D. D. Corkill. Trends in Cooperative Distributed Problem Solving. *IEEE Transactions on Knowledge and Data Engineering*, 1(1):63–83, 1989.

[52] E. H. Durfee and T. A. Montgomery. Coordination as Distributed Search in a Hierarchical Behavior Space. *IEEE Transactions of Systems, Man and Cybernetics*, 21(6):1363–1378, 1991.

[53] Martin Eisenhardt, Wolfgang Müller, and Andreas Henrich. Classifying Documents by Distributed P2P Clustering. In *GI Jahrestagung*, pages 286–291, 2003.

[54] Ronald Fagin, Amnon Lotem, and Moni Naor. Optimal Aggregation Algorithms for Middleware. *Journal of Computer and System Sciences*, 66:614–656, 2003.

[55] Finite element method: Wiki. `http://en.wikipedia.org/wiki/Finite_element_method`.

[56] T. Finin, R. Fritzson, D. McKay, and R. McEntire. KQML as an Agent Communication Language. In *Proceedings of CIKM'94*, pages 456–463, Gaithersburg, MD, USA, 1994. ACM Press.

[57] L.R. Ford and D.R. Fulkerson. *Flows in Networks*. Princeton University Press, 1962.

[58] George Forman and Bin Zhang. Distributed data clustering can be efficient and exact. *SIGKDD Explorations*, 2(2):34–38, 2000.

[59] Ian Foster and Carl Kesselman. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, 2004.

[60] W. Frawley, G. Piatetsky-Shapiro, and C. Matheus. Knowledge Discovery in Databases: An Overview. *AI Magazine*, 1992.

[61] A. Fred and A. Jain. Data Clustering Using Evidence Accumulation. In *Proceedings of ICPR'02*, page 40276, Washington, DC, USA, 2002.

[62] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. Additive Logistic Regression: A Statistical View of Boosting. *The Annals of Statistics*, 38(2):337–374, 2000.

[63] J. Garcia-Luna-Aceves and S. Murthy. A Path-Finding Algorithm for Loop-Free Routing. *IEEE Transactions on Networking*, 5(1):148–160, 1997.

[64] S. Ghiasi, A. Srivastava, X. Yang, and M. Sarrafzadeh. Optimal Energy Aware Clustering in Sensor Networks. *Sensors*, 2:258–269, 2002.

[65] Sukumar Ghosh. *Distributed Systems: An Algorithmic Approach*. CRC press, 2006.

[66] C. Giannella, K. Liu, T. Olsen, and H. Kargupta. Communication Efficient Construction of Deicision Trees Over Heterogeneously Distributed Data. In *Proceedings of ICDM'04*, pages 67–74, Brighton, UK, 2004.

[67] P. Gibbons and S. Tirthapura. Estimating Simple Functions on the Union of Data Streams. In *Proceedings of SPAA'01*, pages 281–291, Crete, Greece, 2001.

[68] Gnutella homepage. `http://www.gnutella.com`.

[69] Claudia V. Goldman and Jeffrey S. Rosenschein. Emergent coordination through the use of cooperative state-changing rules. In *Proceedings of AAAI'94*, pages 408–413, 1994.

[70] Michael B Greenwald and Sanjeev Khanna. Power-Conserving Computation of Order-Statistics over Sensor Networks. In *Proceedings of PODS'04*, pages 275–285, Paris, France, June 2004.

[71] What is Grid ? `http://www.eu-degree.eu/DEGREE/General%\20questions/copy_of_what-is-grid`.

[72] C. Guestrin, P. Bodik, R. Thibaux, M. A. Paskin, and S. Madden. Distributed Regression: an Efficient Framework for Modeling Sensor Network Data. In *Proceddings of IPSN'04*, 2004.

[73] David J. Hand, Heikki Mannila, and Padhraic Smyth. *Principles of Data Mining*. MIT Press, Cambridge, MA, 2001.

[74] Ragib Hasan, Zahid Anwar, William Yurcik, Larry Brumbaugh, and Roy Campbell. A Survey of Peer-to-Peer Storage Techniques for Distributed File Systems. In *Pro-*

ceedings of ITCC'05, pages 205–213, Washington, DC, USA, 2005. IEEE Computer Society.

[75] Thomas Haynes and Sandip Sen. Evolving Behavioral Strategies in Predators and Prey. In *Workshop on Adaptation and Learning in Multiagent Systems part of IJCAI'95*, pages 32–37, Montreal, Quebec, Canada, 20-25 1995.

[76] Wendi Heinzelman, Anantha Chandrakasan, and Hari Balakrishnan. Energy-Efficient Communication Protocol for Wireless Microsensor Networks. In *Proceedings of HICSS'00*, page 10, Hawaii, 2000.

[77] Wendi Heinzelman, Anantha Chandrakasan, and Hari Balakrishnan. An Application-Specific Protocol Architecture for Wireless Microsensor Networks. *IEEE Transactions on Wireless Communications*, 1(4):660–670, 2002.

[78] D. E. Hershberger and H. Kargupta. Distributed Multivariate Regression Using Wavelet-based Collective Data Mining. *Journal of Parallel and Distributed Computing*, 61(3):372–400, 2001.

[79] Carl Hewitt. Viewing Control Structures as Patterns of Passing Messages. *Artificial Intelligence*, 8(3):323–364, 1977.

[80] John H. Holland. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, 1975.

[81] Wolfgang Hoschek, Francisco Javier Jaén-Martínez, Asad Samar, Heinz Stockinger, and Kurt Stockinger. Data Management in an International Data Grid Project. In *Proceedings of the First International Workshop on Grid Computing*, pages 77–90, London, UK, 2000.

[82] Qiang Huang, Helen J. Wang, and Nikita Borisov. Privacy-Preserving Friends Troubleshooting Network. In *Proceedings of NDSS'05*, 2005.

[83] T. Jaakkola. Tutorial on Variational Approximation Methods. *In Advanced Mean Field Methods: Theory and Practice*, 2000.

[84] J.M. Jaffe and F.H. Moss. A Responsive Routing Algorithm for Computer Networks. *IEEE Transactions on Communications*, pages 1758–1762, July 1982.

[85] Mrk Jelasity, Alberto Montresor, and Ozalp Babaoglu. Gossip-based Aggregation in Large Dynamic Networks. *ACM Transactions on Computer Systems (TOCS)*, 23(3):219–252, August 2005.

[86] Finn V. Jensen. *Introduction to Bayesian Networks*. Springer, 1997.

[87] Erik L. Johnson and Hillol Kargupta. Collective, Hierarchical Clustering from Distributed, Heterogeneous Data. In *Revised Papers from Large-Scale Parallel Data Mining, Workshop on Large-Scale Parallel KDD Systems, SIGKDD*, pages 221–244, London, UK, 2000.

[88] Joost homepage. `http://www.joost.com/`.

[89] Michael I. Jordan, Zoubin Ghahramani, Tommi S. Jaakkola, and Lawrence K. Saul. An Introduction to Variational Methods for Graphical Models. *Machine Learning*, 37(2):183–233, November 1999.

[90] JXTA homepage. `https://jxta.dev.java.net/`.

[91] Vana Kalogeraki, Dimitrios Gunopulos, and D. Zeinalipour-Yazti. A Local Search Mechanism for Peer-to-Peer Networks. In *Proceedings of CIKM'02*, pages 300–307, McLean, Virginia, USA, 2002.

[92] H. Kargupta, B. Park, D. Hershbereger, and E. Johnson. Collecttive Data Mining: A New Perspective Towards Distributed Data Mining. In Hillol Kargupta and

Philip Chan, editors, *Advances in Distributed and Parallel Knowledge Discovery*. AAAI/MIT Press, 1999.

[93] H. Kargupta and K. Sivakumar. *Existential Pleasures of Distributed Data Mining. Data Mining: Next Generation Challenges and Future Directions*. AAAI/MIT press, 2004.

[94] Hillol Kargupta and Philip Chan, editors. *Advances in Distributed and Parallel Knowledge Discovery*. MIT Press, 2000.

[95] Hillol Kargupta, Weiyun Huang, Krishnamoorthy Sivakumar, and Erik L. Johnson. Distributed Clustering Using Collective Principal Component Analysis. *Knowledge and Information Systems*, 3(4):422–448, 2001.

[96] Hillol Kargupta, Byung-Hoon Park, and Haimonti Dutta. Orthogonal Decision Trees. *IEEE Trans. on Knowl. and Data Eng.*, 18(8):1028–1042, 2006.

[97] Kazaa homepage. `http://www.kazaa.com`.

[98] Idit Keidar and Assaf Schuster. Want scalable computing?: speculate! *SIGACT News*, 37(3):59–66, 2006.

[99] David Kempe, Alin Dobra, and Johannes Gehrke. Computing Aggregate Information using Gossip. In *Proceedings of FOCS'03*, Cambridge, 2003.

[100] P. M. Khilar and S. Mahapatra. Heartbeat Based Fault Diagnosis for Mobile Ad-Hoc Network. In *Proceedings of IASTED'07*, pages 194–199, Phuket, Thailand, 2007.

[101] Iraklis A. Klampanos and Joemon M. Jose. An Architecture for Information Retrieval over Semi-Collaborating Peer-to-Peer Networks. In *Proceedings of SAC'04*, pages 1078–1083, Nicosia, Cyprus, 2004.

[102] M. Klusch, S. Lodi, and G. L. Moro. Distributed Clustering Based on Sampling Local Density Estimates. In *Proceedings of IJCAI'03*, pages 485–490, Mexico, August 2003.

[103] J. Kotecha, V. Ramachandran, and A. Sayeed. Distributed Multi-target Classification in Wireless Sensor Networks. *IEEE Journal of Selected Areas in Communications (Special Issue on Self-Organizing Distributed Collaborative Sensor Networks)*, 23(4):703–713, 2005.

[104] Wojtek Kowalczyk, Márk Jelasity, and A. E. Eiben. Towards Data Mining in Large and Fully Distributed Peer-to-Peer Overlay Networks. In *Proceedings of BNAIC'03*, pages 203–210, University of Nijmegen, 2003.

[105] D. Krivitski, A. Schuster, and R. Wolff. A Local Facility Location Algorithm for Large-Scale Distributed Systems. *Journal of Grid Computing*, 5(4):361–378, 2007.

[106] Fabian Kuhn, Thomas Moscibroda, and Roger Wattenhofer. What Cannot Be Computed Locally! In *Proceedings of PODC'04*, pages 300–309, St. John's, Newfoundland, Canada, 2004.

[107] A. Kulakov and D. Davcev. Data Mining in Wireless Sensor Networks Based on Artificial Neural-Networks Algorithms. In *Proceedings of 1st International Workshop on Data Mining in Sensor Networks*, pages 10–17, 2005.

[108] S. Kutten and D. Peleg. Fault-Local Distributed Mending. In *Proceedings of PODC'95*, pages 20–27, Ottawa, Canada, August 1995.

[109] V. R. Lesser and D. D. Corkill. Functionally Accurate Cooperative Distributed Systems. *IEEE Transactions on Systems, Machines and Cybernetics*, SMC-11(1):81–96, 1981.

[110] N. Li, J. Hou, and L. Sha. Design and Analysis of an MST-Based Topology Control Algorithm. *IEEE Transactions on Wireless Communications*, 4(3):1195–1205, 2005.

[111] Stephanie Lindsey, Cauligi Raghavendra, and Krishna M. Sivalingam. Data Gathering Algorithms in Sensor Networks Using Energy Metrics. *IEEE Transactions on Parallel and Distributed Systems*, 13(9):924–935, 2002.

[112] N. Linial. Locality in Distributed Graph Algorithms. *SIAM Journal of Computing*, 21:193–201, 1992.

[113] LionShare: Secure P2P file sharing application for higher education. `http://lionshare.psu.edu/`.

[114] Michael L. Littman. Markov Games as a Framework for Multi-Agent Reinforcement Learning. In *Proceedings of ICML'94*, pages 157–163, New Brunswick, NJ, 1994.

[115] K. Liu, K. Bhaduri, K. Das, P. Nguyen, and H. Kargupta. Client-side Web Mining for Community Formation in Peer-to-Peer Environments. *SIGKDD Explorations*, 8(2):11–20, 2006.

[116] Ping Luo, Hui Xiong, Kevin Lü, and Zhongzhi Shi. Distributed Classification in Peer-to-Peer Networks. In *Proceedings of SIGKDD'07*, pages 968–976, 2007.

[117] Qin Lv, Pei Cao, Edith Cohen, Kai Li, and Scott Shenker. Search and Replication in Unstructured Peer-to-Peer Networks. In *Proceedings of ICS '02*, pages 84–95, 2002.

[118] A. Manjhi, V. Shkapenyuk, K. Dhamdhere, and C. Olston. Finding (Recently) Frequent Items in Distributed Data Streams. In *Proceedings of ICDE'05*, pages 767–778, Washington, DC, USA, 2005.

[119] Mortada Mehyar, Demetri Spanos, John Pongsajapan, Steven H. Low, and Richard

Murray. Distributed Averaging on Peer-to-Peer Networks. In *Proceedings of CDC'05*, Spain, 2005.

[120] Srujana Merugu and Joydeep Ghosh. A Distributed Learning Framework for Heterogeneous Data Sources. In *Proceedings of KDD'05*, pages 208–217, 2005.

[121] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller. Equations of State Calculations by Fast Computing Machines. *Journal of Chemical Physics*, 21:1087–1092, 1953.

[122] Sebastian Michel, Peter Triantafillou, and Gerhard Weikum. KLEE: a framework for distributed top-k query algorithms. In *Proceedings of VLDB'05*, pages 637–648, Trondheim, Norway, 2005.

[123] Ingo Mierswa, Katharina Morik, and Michael Wurst. Collaborative Use of Features in a Distributed System for the Organization of Music Collections. In Shen, Shepherd, Cui, and Liu, editors, *Intelligent Music Information Systems: Tools and Methodologies*, pages 147–175. Information Science Reference, 2007.

[124] Dejan S. Milojicic, Vana Kalogeraki, Rajan Lukose, Kiran Nagaraja, Jim Pruyne, Bruno Richard, Sami Rollins, and Zhichen Xu. Peer-to-peer computing.

[125] Marvin Minsky. *The Society of Mind*. New York: Simon and Schuster, 1986.

[126] Stephen B Montgomery, Tony Fu, Jun Guan, Keven Lin, and Steven J M Jones. An Application of Peer-to-Peer Technology to the Discovery, Use and Assessment of Bioinformatics Programs. *Nature Methods*, page 563, 2005.

[127] Yishay Mor and Jeffrey S. Rosenschein. Time and the Prisoner's Dilemma. In *Proceedings of the First International Conference on Multiagent Systems.*, pages 276–282, Menlo park, California, June 1995.

[128] Alexandros Moukas, Konstantinos Chandrinos, and Pattie Maes. Trafficopter: A Distributed Collection System for Traffic Information. In *Proceedings of the Second International Workshop on Cooperative Information Agents II, Learning, Mobility and Electronic Commerce for Information Discovery on the Internet*, pages 33–43, London, UK, 1998.

[129] S. Mukherjee and H. Kargupta. Distributed Probabilistic Inferencing in Sensor Networks using Variational Approximation. *Journal of Parallel and Distributed Computing (JPDC) (in press)*, 2007.

[130] Sourav Mukherjee. A Variational Approach Towards Distributed Data Mining. Master's thesis, University of Maryland, baltimore County, May 2007.

[131] Sourav Mukherjee and Hillol Kargupta. Communication-efficient multivariate regression from heterogeneously distributed data using variational approximation. In *Communication*, 2007.

[132] Kevin Murphy. A Variational Approximation for Bayesian Networks with Discrete and Continuous Latent. In *Proceedings of UAI'99*, pages 457–46, San Francisco, CA, 1999. Morgan Kaufmann.

[133] Moni Naor and Larry Stockmeyer. What Can be Computed Locally? In *Proceedings of STOC'93*, pages 184–193, 1993.

[134] Napster homepage. `http://www.napster.com`.

[135] Luis Ernesto Navarro-Serment, John Dolan, and Pradeep Khosla. Optimal Sensor Placement for Cooperative Distributed Vision. In *Proceedings of ICRA'04*, pages 939–944, April 2004.

[136] Wolfgang Nejdl, Wolf Siberski, Uwe Thaden, and Wolf-Tilo Balke. Top-k Query

Evaluation for Schema-Based Peer-to-Peer Networks. In *Proceedings of ISWC'04*, pages 137–151, Hiroshima, Japan, 2004.

[137] Wolfgang Nejdl, Boris Wolf, Changtao Qu, Stefan Decker, Michael Sintek, Ambjörn Naeve, Mikael Nilsson, Matthias Palmér, and Tore Risch. EDUTELLA: A P2P Networking Infrastructure Based on RDF. In *Proceedings of WWW'02*, pages 604–615, Honolulu, Hawaii, USA, 2002.

[138] J.A. Nelder and R. Mead. A Simplex Method for Function Minimization. *The Computer Journal*, 7:308–313, 1965.

[139] John Von Neumann. *Theory of Self-Reproducing Automata*. University of Illinois Press, Champaign, IL, USA, 1966.

[140] H. Penny Nii. Blackboard Systems, Part One: The Blackboard Model of Problem Solving and the Evolution of Blackboard Architectures. *AI Magazine*, 7(2):38–53, 1986.

[141] Todd Olsen. Distributed Decision Tree Learning From Multiple Heterogeneous Data Sources. Master's thesis, University of Maryland, Baltimore County, Baltimore. Maryland, October 2006.

[142] C. Olston, J. Jiang, and J. Widom. Adaptive Filters for Continuous Queries over Distributed Data Streams. In *Proceedings of SIGMOD'03*, pages 563–574, San Diego, California, 2003.

[143] Ömer Egecioglu and Hakan Ferhatosmanoglu. Dimensionality Reduction and Similarity Computation by Inner Product Approximations. In *Proceedings of CIKM'00*, pages 219–226, New York, 2000.

[144] P2P Wikipedia homepage. `http://en.wikipedia.org/wiki/Peer-to-peer`.

[145] Noam Palatin, Arie Leizarowitz, Assaf Schuster, and Ran Wolff. Mining for Misconfigured Machines in Grid Systems. In *Proceedings of SIGKDD'06*, pages 687–692, Philadelphia, PA, USA, 2006.

[146] Themistoklis Palpanas, Dimitris Papadopoulos, Vana Kalogeraki, and Dimitrios Gunopulos. Distributed Deviation Detection in Sensor Networks. *ACM SIGMOD Record*, 32(4):77–82, December 2003.

[147] B. Park, R. Ayyagari, and H. Kargupta. A Fourier Analysis-Based Approach to Learn Classifier from Distributed Heterogeneous Data. In *Proceedings of SDM'01*, Chicago, IL, April 2001.

[148] B. Park, H. Kargupta, E. Johnson, E. Sanseverino, D. Hershberger, and L. Silvestre. Distributed, Collaborative Data Analysis from Heterogeneous Sites Using a Scalable Evolutionary Technique. *Applied Intelligence*, 16(1):19–42, 2002.

[149] David Peleg. *Distributed Computing: a Locality-Sensitive Approach*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2000.

[150] T. Pham, D. S. Scherber, and H. C. Papadopoulos. Distributed Source Localization Algorithms for Acoustic Ad-hoc Sensor Networks. In *Proceedings of Sensor Array and Multichannel Signal Processing Workshop*, pages 613–617, Barcelona, Spain, 2004.

[151] PlanetLab homepage. `http://www.planet-lab.org/`.

[152] PodPlaylist: Sharing iTune Playlists. `http://www.podplaylist.com/`.

[153] J.B. Predd, S.R. Kulkarni, and H.V. Poor. Distributed Kernel Regression: An Algorithm for Training Collaboratively. *arXiv.cs.LG archive*, 2006.

[154] J. R. Quinlan. Induction of Decision Trees. *Machine Learning*, 1(1):81–106, 1986.

[155] J. Ross Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kauffman, 1993.

[156] Michael Rabbat and Robert Nowak. Distributed Optimization in Sensor Networks. In *Proceedings of IPSN'04*, pages 20–27, Berkeley, California, USA, 2004.

[157] Predrag Radivojac, Uttara Korad, K. M. Sivalingam, and Zoran Obradovic. Learning from Class-Imbalanced Data in Wireless Sensor Networks. In *Proceedings of VTC'03 Fall*, pages 3030 – 3034, Orlando, Florida, 2003.

[158] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Schenker. A Scalable Content-Addressable Network. In *Proceedings of SIGCOMM'01*, pages 161–172, 2001.

[159] S. E. Robertson and K. S. Jones. Relevance Weighting of Search Terms. *Journal of the American Society for Information Science*, 27:129–146, 1976.

[160] Antony Rowstron and Peter Druschel. Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems. In *Proceedings of IFIP/ACM Middleware'01*, pages 329–350, 2001.

[161] G. Salton, A.Wang, and C. Yang. A Vector Space Model for Information Retrieval. *Journal of the American Society for Information Science*, 18:613–620, 1975.

[162] Andrea Schaerf, Yoav Shoham, and Moshe Tennenholtz. Adaptive Load Balancing: A Study in Multi-Agent Learning. *Journal of Artificial Intelligence Research*, 2:475–500, 1995.

[163] D. S. Scherber and H. S. Papadopoulos. Distributed Computation of Averages Over ad hoc Networks. *IEEE Journal on Selected Areas in Communications*, 23(4):776–787, 2005.

[164] A. Schuster and R. Wolff. Communication-Efficient Distributed Mining of Association Rules. *Data Mining and Knowledge Discovery*, 8(2):171–196, 2004.

[165] ScienceNet Peer-to-Peer Secarch Engine homepage. `http://liebel.fzk.de/collaborations/sciencenet-search-engine-based-on-yacy-p2p-technology`.

[166] Sandip Sen. Multiagent Systems: Milestones and New Horizons. *Trends in Cognitive Science*, 1(9):334–339, 1997.

[167] Sandip Sen, Shounak Roychowdhury, and Neeraj Arora. Effects of Local Information on Group Behavior. In *BIBICMAS*, pages 315–321, Menlo Park, CA, 1996.

[168] I. Sharfman, A. Schuster, and D. Keren. A Geometric Approach to Monitoring Threshold Functions over Distributed Data Streams. In *Proceedings of SIGMOD'06*, pages 301–312, Chicago, Illinois, June 2006.

[169] Onn Shehory and Sarit Kraus. Methods for Task Allocation via Agent Coalition Formation. *Artificial Intelligence*, 101(1-2):165–200, 1998.

[170] ShuMing Shi, Jin Yu, GuangWen Yang, and DingXing Wang. Distributed Page Ranking in Structured P2P Networks. In *Proceedings of ICPP'03*, page 179, Los Alamitos, CA, USA, 2003.

[171] Ka Cheung Sia. P2P Information Retrieval: A Self-Organizing Paradigm. Technical report, 2002. `http://www.cse.cuhk.edu.hk/~kcsia/tp3.pdf`.

[172] Skype homepage. `http://www.skype.com/`.

[173] R. G. Smith. The Contract Net Protocol: High-level Communication and Control in a Distributed Problem Solver. *Distributed Artificial Intelligence*, pages 357–366, 1988.

[174] SopCast homepage. `http://www.sopcast.org/`.

[175] Ion Stoica, Robert Morris, David Karger, Frans Kaashoek, and Hari Balakrishnan. Chord: A Scalable Peer-To-Peer Lookup Service for Internet Applications. In *Proceedings of SIGCOMM'01*, pages 149–160, 2001.

[176] Salvatore J. Stolfo, Andreas L. Prodromidis, Shelley Tselepis, Wenke Lee, Dave W. Fan, and Philip K. Chan. JAM: Java Agents for Meta-Learning over Distributed Databases. In *Proceedings of KDD'97*, pages 74–81, Newport Beach, CA, 1997.

[177] Peter Stone. *Layered Learning in Multi-Agent Systems*. PhD thesis, Carnegie Mellon University, December 1998.

[178] Peter Stone and Manuela M. Veloso. Multiagent Systems: A Survey from a Machine Learning Perspective. *Autonomous Robots*, 8(3):345–383, 2000.

[179] Alexander Strehl and Joydeep Ghosh. Cluster Ensembles — A Knowledge Reuse Framework for Combining Multiple Partitions. *Journal of Machine Learning Research*, 3:583–617, 2003.

[180] Torsten Suel, Chandan Mathur, Jo wen Wu, Jiangong Zhang, Alex Delis, Mehdi Kharrazi, Xiaohui Long, and Kulesh Shanmugasundaram. ODISSEA: A Peer-to-Peer Architecture for Scalable Web Search and Information Retrieval. In *WebDB'03*, pages 67–72, 2003.

[181] Katia Sycara, Anandee Pannu, Mike Williamson, Dajun Zeng, and Keith Decker. Distributed Intelligent Agents. *IEEE Expert: Intelligent Systems and Their Applications*, 11(6):36–46, 1996.

[182] D. Talia and D. Skillicorn. Mining Large Data Sets on Grids: Issues and Prospects. *Computing and Informatics*, 21(4):347–362, 2002.

[183] D. Talia and P. Trunfio. Toward a Synergy Between P2P and Grids. *IEEE Internet Computing*, 7(4):94–95, August 2003.

[184] Ming Tan. Multi-agent Reinforcement Learning: Independent vs. Cooperative Agents. *Readings in agents*, pages 487–494, 1998.

[185] Pang-Ning Tan, Michael Steinbach, and Vipin Kumar. *Introduction to Data Mining*. Addison-Wesley, 2006.

[186] Chunqiang Tang, Zhichen Xu, and Sandhya Dwarkadas. Peer-to-Peer Information Retrieval using Self-organizing Semantic Overlay Networks. In *Proceedings of SIGCOMM'03*, pages 175–186, Karlsruhe, Germany, 2003.

[187] C. Tempich, A. Löser, and J. Heizmann. *Community Based Ranking in Peer-to-Peer Networks*, volume 3761, chapter Lecture Notes in Computer Science, pages 1261–1278. Springer-Verlag GmbH, October 2005.

[188] Tranche homepage. `http://tranche.proteomecommons.org/`.

[189] Kagan Tumer and Joydeep Ghosh. Robust combining of disparate classifiers through order statistics. *Pattern Analysis and Applications*, 5:189–200, 2001.

[190] Ubistorage homepage. `http://osa.inria.fr/wiki/Developments/ModelingOfPeer-to-peerFileStorageSystems`.

[191] Vehicular Lab: UCLA Compter Science Department. `http://www.vehicularlab.org/`.

[192] V. Vapnik. *Statistical Learning Theory*. Wiley-Interscience, 1998.

[193] Grid Page: Wiki. `http://en.wikipedia.org/wiki/Grid_computing`.

[194] Ian H. Witten and Eibe Frank. *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann, 2005.

[195] R. Wolff, K. Bhaduri, and H. Kargupta. Local L2 Thresholding Based Data Mining in Peer-to-Peer Systems. In *Proceedings of SDM'06*, pages 428–439, 2006.

[196] R. Wolff and A. Schuster. Association Rule Mining in Peer-to-Peer Systems. *IEEE Transactions on Systems, Man and Cybernetics - Part B*, 34(6):2426 – 2438, December 2004.

[197] Y. Xing, M. G. Madden, J. Duggan, and G. J. Lyons. Distributed Regression for Heterogeneous Data Sets. *Lecture Notes in Computer Science*, 2810:544–553, 2003.

[198] Beverly Yang and Hector Garcia-Molina. Improving Search in Peer-to-Peer Networks. In *Proceedings of ICDCS'02*, pages 5–14, Washington, DC, USA, 2002. IEEE Computer Society.

[199] Wai Gen Yee and Ophir Frieder. On Search in Peer-to-Peer File Sharing Systems. In *Proceedings of SAC'05*, pages 1023–1030, Santa Fe, New Mexico, 2005.

[200] O. Younis and S. Fahmy. Heed: A hybrid, Energy-Efficient, Distributed Clustering Approach for ad-hoc Sensor Networks. *IEEE Transactions on Mobile Computing*, 3(4):258–269, 2004.

[201] Mohammed Javeed Zaki. Parallel and Distributed Association Mining: A Survey. *IEEE Concurrency*, 7(4):14–25, 1999.

[202] Mohammed Javeed Zaki. Parallel and Distributed Data Mining: An Introduction. In *Revised Papers from Large-Scale Parallel Data Mining, Workshop on Large-Scale Parallel KDD Systems, SIGKDD*, pages 1–23, London, UK, 2000.

[203] Ben Y. Zhao, Ling Huang, Jeremy Stribling, Sean C. Rhea, Anthony D. Joseph, and John D. Kubiatowicz. Tapestry: A Resilient Global-Scale Overlay for Service Deployment. *IEEE Journal On Selected Areas In Communications*, 22(1):41–53, 2004.

[204] Feng Zhao and Leonidas Guibas. *Wireless Sensor Networks: An Information Processing Approach*. Morgan Kaufmann, 2004.

[205] J. Zhao, R. Govindan, and D. Estrin. Computing Aggregates for Monitoring Wireless Sensor Networks. In *Proceedings of the First IEEE International Workshop on Sensor Network Protocols and Applications*, pages 139–148, 2003.

[206] Alice X. Zheng, Jim Lloyd, and Eric Brewer. Failure Diagnosis Using Decision Trees. In *Proceedings of ICAC '04*, pages 36–43, Washington, DC, USA, 2004.