Anomaly Detection Techniques for Ad Hoc Networks

Chaoli Cai

Western Michigan University

Follow this and additional works at: http://scholarworks.wmich.edu/dissertations

Part of the Computer Sciences Commons

Recommended Citation
http://scholarworks.wmich.edu/dissertations/652

This Dissertation-Open Access is brought to you for free and open access by the Graduate College at ScholarWorks at WMU. It has been accepted for inclusion in Dissertations by an authorized administrator of ScholarWorks at WMU. For more information, please contact maira.bundza@wmich.edu.
ANOMALY DETECTION TECHNIQUES FOR AD HOC NETWORKS

by

Chaoli Cai

A Dissertation
Submitted to the
Faculty of The Graduate College
in partial fulfillment of the
requirements for the
Degree of Doctor of Philosophy
Department of Computer Science
Advisor: Ajay Gupta, Ph.D.

Western Michigan University
Kalamazoo, Michigan
December 2009
ANOMALY DETECTION TECHNIQUES FOR AD HOC NETWORKS

Chaoli Cai, Ph.D.

Western Michigan University, 2009

Anomaly detection is an important and indispensable aspect of any computer security mechanism. Ad hoc and mobile networks consist of a number of peer mobile nodes that are capable of communicating with each other absent a fixed infrastructure. Arbitrary node movements and lack of centralized control make them vulnerable to a wide variety of unknown and known attacks from inside as well as from outside. In this dissertation we propose two efficient statistical techniques for anomaly detection for these networks.

We present a mobility-pattern-based (MPB) anomaly detection algorithm that can identify abnormal pattern behavior of nodes in mobile networks. MPB characterizes the mobility profile of a node by a Multi-Leaf tree structure in which each node corresponds to a possible destination cluster. Through data mining and fuzzy logic techniques, a normal mobility profile is generated during the training process, and abnormal patterns are distinguished from the normal during testing. Statistical simulations demonstrate that proposed MPB algorithm achieves reasonably low false alarm rates (FAR) and sufficiently high detection rates (DR).

In order to take into account incomplete testing samples and the interaction among multiple features, we present BANBAD – a technique using Belief Networks and Bayesian inference. BANBAD identifies abnormal behavior in any feature, e.g., inappropriate energy consumption of a node in the network. By applying structure learning techniques to the training dataset, it extracts the dependencies among
relevant features and represents them by a directed acyclic graph. Probability distributions are associated with the nodes (i.e., features) and edges of the graph. BANBAD maintains this belief network as a dynamic, updated normal profile of feature behaviors and then uses a specific Bayesian inference algorithm to detect abnormal behavior in testing data. Our technique works especially well in ad hoc networks but is applicable to other networks including wireless and sensor networks. The proposed method bounds FAR at a predefined threshold and maximizes DR. Experimental results demonstrate excellent performance for synthetic as well as real datasets. The real datasets are taken from Intel Lab Data (lab environment monitored by the sensors) and UMASS Trace Repository (users’ laptop usage).
NOTE TO USERS

This reproduction is the best copy available.

UMI
I could never have reached the heights or explored the depths without the support, guidance and efforts of many people who have contributed to the production of my dissertation.

First, I am grateful to — and would like to sincerely thank — my advisor and mentor, Dr. Ajay Gupta for your guidance and insight. It has been an honor for me to work with you during the past three years at Western Michigan University. What I value most was the opportunity to absorb not only your insights into the specifics of my work, but also a fundamental approach to research that emphasizes rigor and first principles. I hope this dissertation will go some way towards expressing my thanks.

Second, this dissertation would not have been possible without the expert guidance of Dr. Paul who is my committee member. Your statistical background and specialty always give me so many inspirations, and your oral and written comments are extremely perceptive, helpful and appropriate. I appreciate your extremely huge support.

I would like to thank another dissertation committee member: Dr. Leszek Lilien, for your feedback, and tremendous support.

I also would like to thank my previous committee members: Dr. Ala Al-Fuqaha, Dr. Mohsen Guizani, Dr. Dionysios Kountains, and Dr. Song Ci, for their support to my previous research work.

I am also indebted to Aakash Gupta for your invaluable help during the earlier stages of this study. I greatly value the support I received from all of my faithful friends: Lezheng Liu, Peng Huang, Song Gao, Wenhui Yuan, Jing Cai, Xiaoming

I would like to extend my gratitude to my family and most importantly my parents. The most whole-hearted words can not express my thanks to them for their kindness, persistence and self-sacrifices. Throughout my life, their faith and love has given me the strength and courage to believe in myself. I also would like to thank my sister who always encourages me, hope you have a happy family and wonderful future in Shanghai.

Finally, I am grateful to my loving and ever supportive fiancée, Dr. Xiaofan Cai. To my fiancée, I would say many thanks for being the inspiration of my life that enabled me to complete this work. At the same time, I am very sorry that you left this world so early. You always live in my life, and I would share my every success with you. Also, I would like to express my respect and admiration to your family and all the friends who helped you during that hardship period. Thanks all.

Chaoli Cai
# TABLE OF CONTENTS

ACKNOWLEDGMENTS ........................................................................................................ ii

LIST OF TABLES ................................................................................................................ vii

LIST OF FIGURES ............................................................................................................... viii

CHAPTER

I. INTRODUCTION .............................................................................................................. 1
   1.1 Motivation .................................................................................................................. 1
   1.2 Problem Definitions ................................................................................................. 3
       1.2.1 Mobility-pattern-based (MPB) anomaly detection ........................................ 3
       1.2.2 Belief networks based anomaly detection ..................................................... 5
   1.3 Results ..................................................................................................................... 6
   1.4 Dissertation Organization ...................................................................................... 7

II. RELATED WORK AND BACKGROUND ......................................................................... 9
   2.1 Intrusion Prevention Techniques ............................................................................ 12
   2.2 Intrusion Detection Techniques ............................................................................ 13
   2.3 Belief Networks .................................................................................................... 17
   2.4 Structure Learning Techniques ............................................................................ 17

III. ANOMALY DETECTION FOR MOBILE NETWORKS .................................................. 19
   3.1 Model Description ................................................................................................. 19
       3.1.1 Network model ............................................................................................... 19
       3.1.2 Mobility model ................................................................................................ 19
   3.2 MPB Anomaly Detection Algorithm ..................................................................... 20
Table of Contents—continued

CHAPTER

3.2.1 Pattern definition ................................................................. 20
3.2.2 Data mining and fuzzy logic approaches................................. 20
3.2.3 The distance measure............................................................ 22
3.2.4 Multi-leaf tree structure ......................................................... 22
3.2.5 The framework................................................................. 24
3.2.6 Overhead of MPB anomaly detection algorithm ..................... 25
3.3 Simulation Results ................................................................. 27

IV. CENTRALIZED BANBAD ANOMALY DETECTION ALGORITHM ......................................................... 32

4.1 Assumptions ............................................................................. 32
4.2 Belief Networks and BANBAD Preliminaries .................................. 34
4.3 The Directed Acyclic Graph (DAG) Model................................... 38
4.4 Training and Testing Processes .................................................. 39
4.5 Range Settings ........................................................................ 41
4.6 Belief Computation ................................................................ 42

V. SIMULATION RESULTS OF CENTRALIZED BANBAD ....................................... 47

5.1 Synthetic Datasets ...................................................................... 47
5.1.1 Normal distribution ................................................................. 47
5.1.2 Gamma distribution ................................................................. 55
5.1.3 Non-linear relationship features .............................................. 56
5.2 Effect of Categorization ............................................................... 58

VI. BANBAD PERFORMANCE ON REAL DATASETS ........................................ 61
Table of Contents—continued

CHAPTER

6.1 Wireless Sensor Network (WSN) .................................................. 61
6.2 Wireless Network (WN) ............................................................. 63

VII. BANBAD TOOLKIT ...................................................................... 66
7.1 Purpose ....................................................................................... 66
7.2 Organization .............................................................................. 66
7.3 HW/SW Platforms ..................................................................... 68
7.4 Installation .................................................................................. 68
7.5 Main Functions/Classes of BANBAD Software ......................... 69
7.6 Tutorial ....................................................................................... 71

VIII. SUMMARY AND FUTURE WORK ........................................... 87

APPENDICES

A. BANBAD Toolkit Source Code ...................................................... 90

BIBLIOGRAPHY ............................................................................... 130
LIST OF TABLES

5.1: Categories of 6 features for Scheme 1 ................................................................. 48
6.1: Range of 4 features of WSN .............................................................................. 62
6.2: Categories of 4 features of WSN ........................................................................ 62
6.3: Anomaly sample # and performance of WSN ..................................................... 62
6.4: Range of 6 features of WN ................................................................................ 64
6.5: Categories of 6 features of WN .......................................................................... 64
6.6: Anomaly sample # and performance of WN ....................................................... 64
6.7: Anomaly sample # and performance of WN ....................................................... 64
LIST OF FIGURES

1.1: Threat model of MPB
2.1: Classifications of IDPS
3.1: Cluster classification algorithm
3.2: Multi-leaf tree structure
3.3: General framework of MPB anomaly detection algorithm
3.4: A specific route \((C_1\rightarrow C_{11} \rightarrow C_{111})\) of three hops after the training process
3.5: Performance at different velocity ranges
3.6: Fixed mobility pattern threshold
4.1: The chain application model displaying the dependencies of various features of a node
4.2: A more complete DAG application model representing the various factors that affect energy consumption of a node
4.3: The training process of BANBAD
4.4: The testing process of BANBAD
4.5: Belief computation during the training process
4.6: Testing process of BANBAD for the chain \(D \rightarrow E \rightarrow R\)
5.1: Structure of the profile learnt by PC
5.2: False alarm rate and detection rate for all features (normal distribution)
5.3: False alarm rate of feature 4
5.4: Detection rate of feature 4
5.5: FAR bound versus DR for State 2 of feature 4
5.6: False alarm rate and detection rate for all features (gamma distribution)
List of Figures—continued

5.7: False alarm rate and detection rate for all features having non-linear relationship .................................................................57

5.8: Effect of categorization of feature 4 .................................................................................................................................59

5.9: A reason that BANBAD is not applicable for anomaly detection for individual states ..........................................................60

7.1: General structure of the BANBAD toolkit .......................................................................................................................66

7.2: Organization of the BANBAD toolkit .............................................................................................................................67

7.3: Function of “categorization” ...........................................................................................................................................69

7.4: Function of “causalReasoningComputation” ....................................................................................................................70

7.5: Function of “diagnosticReasoningComputation” ...............................................................................................................70

7.6: Function of “beliefComputation” ....................................................................................................................................71

7.7: Waiting for loading raw dataset .......................................................................................................................................72

7.8: Waiting for categorization ..................................................................................................................................................73

7.9: Categorization ..................................................................................................................................................................74

7.10: Waiting for structure learning .........................................................................................................................................75

7.11: Structure learned ..............................................................................................................................................................76

7.12: Waiting for computing and displaying causal reasoning ..................................................................................................77

7.13: Display causal reasoning .....................................................................................................................................................78

7.14: Display diagnostic reasoning .............................................................................................................................................79

7.15: Display belief ......................................................................................................................................................................80

7.16: Normal range of the target feature ..................................................................................................................................81

7.17: Waiting to start testing process .........................................................................................................................................81

7.18: Compute and display belief values of the target feature for FAR .....................................................................................82
List of Figures—continued

7.19: Compute and display belief values of the target feature for DR ......................83
7.20: False alarm rate of the running example ..........................................................84
7.21: Detection rate of the running example .............................................................85
7.22: Final screen when the BANBAD toolkit completes one run .........................86
CHAPTER I

INTRODUCTION

1.1 Motivation

Ad hoc and mobile networks consist of a number of peer mobile nodes that are capable of communicating with each other absent a fixed infrastructure. However, arbitrary node movements and lack of centralized control make them vulnerable to a wide variety of attacks from inside as well as from outside. Therefore, providing effective security protection is important to ensure the continued viability of Ad hoc and mobile networks in a variety of pursuits.

In general, two complementary approaches exist to protect a system: prevention and detection. Intrusion prevention techniques, such as encryption and authentication, attempt to deter and block attackers. Unfortunately, prevention techniques can only reduce intrusions, not completely eliminate them [1-2]. Despite the amount or quality of intrusion prevention measures, an intelligent attacker can exploit a single security hole to break into a system. Nothing is absolutely secure. Therefore, intrusion detection systems (IDSs) are indispensable for a reliable system. They serve as the important secondary line of defense.

Intrusion detection can be based either on detecting misuses or detecting anomalies. A misuse-based detection technique checks potential security breaches
against known attack signatures and system vulnerabilities. If it finds a match, an alarm is generated. Since it is impossible to know all future attacks—or attack patterns—in advance, misuse detection techniques are not effective in detecting new or unknown attacks. Given the constantly evolving nature of security breaches, anomaly-based techniques are needed. An anomaly-based detection technique models normal behavior by creating profiles of system and node states during the training process. During the testing process, it compares deviations from the normal profiles to determine whether a deviation is significant. If so, an alarm is triggered. Therefore, anomaly detection can check a whole host of different and new types of attacks. While misuse detection may be more efficient, anomaly detection is more comprehensive. In security, comprehensiveness is best. Anything less leaves systems open for attack.

Efficiently establishing and maintaining profiles for nodes is crucial for anomaly detection. Unfortunately, the mobility of nodes inherent in ad hoc network makes profile generation difficult. Furthermore, because of the ad hoc nature of the network, availability of complete data is often not possible; therefore a technique handling incomplete data is desired. A technique based on belief networks is proposed in this dissertation to address these issues. Our technique will be referred as BANBAD - Belief Network Based Anomaly Detection.

In Chapter III, we propose a novel approach to construct the normal mobility profile of a node, from which an efficient mobility-pattern-based (MPB) anomaly detection algorithm is designed. The sequence of the clusters and mobility pattern
strings traversed by a node is used as the feature. When an intrusion occurs, the attacker tends to have a different mobility pattern. We can detect anomaly by comparing it with the normal mobility profile of that node.

MPB technique facilitates detection of anomaly in a single feature such as the mobility. However, in a typical system multiple features interact with each other and the question of designing a technique that can handle multiple features arises. We thus propose BANBAD that has the ability to handle multiple features anomaly detection starting from chapter IV. One of the main purposes of our research is to try to address the shortcomings (please see section 2.2 for specific details), to the best of our capabilities, in current anomaly detection methods to create an affordable, efficient, and effective anomaly detection method for ad hoc networks. The key improvements of the proposed BANBAD is the ability to obtain high detection rate (DR) while decreasing false alarm rate (FAR), be able to bound FAR, and handle incomplete samples.

1.2 Problem Definitions

1.2.1 Mobility-pattern-based (MPB) anomaly detection

Since mobility is the most typical and most crucial feature of a node in mobile networks, we focus on the detection of subtle difference by handling close points and approximate mobility patterns of mobile networks. Our algorithm attempts to detect one of the most important active attacks -- the deviation from the normal mobility
patterns from a starting point to any subsequent point. In this context, point is not a single geographical point; instead it implies a cluster or a region. Consider Figure 1.1 that shows few possible paths from a point $a_i$ in region A to a point $b_j$ in region B:

There are two geographical areas, A and B as well as four routes, R1: $a_i \rightarrow o \rightarrow b_j$, R2: $a_i \rightarrow o' \rightarrow b_j$, R3: $a_i \rightarrow m \rightarrow b_j$ and R4: $a_i \rightarrow m' \rightarrow b_{j'}$, which all have two hops starting from starting point $a_i$ to destination point, $b_j$ or $b_{j'}$. We assume $b_j$ and $b_{j'}$ are very close and the angle $\alpha = \angle m \ a_i m'$ is very small.

Case 1: Suppose R1 is the normal route, then the other 3 routes should be treated as anomalies. Note $b_j$ and $b_{j'}$ are close and R2, R3 & R4 can be reached.

Figure 1.1: Threat model of MPB.
within the same time period (2 hops). Even if R2 has the same pattern distributions as R1, we still need to classify it as an anomaly route.

**Case 2:** Suppose R3 is the normal route. Based on the obvious deviations, it’s still easy to classify R1 and R2 as anomalies, but what about R4?

2.1. If the situation occurs in the training process, should we treat R3 and R4 as 2 distinct routes or integrate them as just 1 route? What are the shortcomings of the former solution? And if we choose the latter solution, what classifies the set of “close” normal routes?

2.2. If the situation occurs during the testing process, can we classify R4 as the normal route? Or as an anomaly? Is there any quantified approach that exists to solve such problems?

Clearly, case 2 is the most challenging issue. It tries to address the problem with many closer points and mobility patterns in an observed cluster set. Our proposed algorithm attempts to handle close proximity of mobility patterns.

1.2.2 Belief networks based anomaly detection

MPB anomaly detection focuses on one feature (mobility) anomaly detection; however, mobility is not the only feature. In ad hoc and mobile networks, there are many crucial features, e.g., energy consumption, local computation, response time, etc. Intrusions can occur in any of these features at any time. Also, due to the system/network failure, the data we collected may not be always complete. Therefore, the following questions arise: Can we detect anomaly in the testing data? Can we
detected anomaly in a specific feature? Can we detect anomaly if some data is missing?

In order to handle the interactions among multiple features and missing data, we propose a Belief Network Based Anomaly Detection (BANBAD) technique.

1.3 Results

The MPB anomaly detection algorithm is effective for mobile networks. Each node's normal mobility profile is modeled as a Multi-Leaf tree structure. Clusters, generated through data mining, and the corresponding pattern strings, generated through fuzzy logic, are the two fundamental elements of the Multi-Leaf tree structure. Our MPB algorithm is then developed to detect potential masquerade attacks. Simulation results demonstrate that our proposed algorithm can achieve desirable performance in terms of both the false alarm rate and the detection rate for nodes with regular movement behaviors through a fine tuning of the design parameter \( \text{threshold} \). Efficient fine-tuning of this algorithm provides success and, subsequently, a basis for better anomaly detection in all mobile networks.

BANBAD is an efficient anomaly detection technique which has the potential to achieve high detection rates while reducing false alarm rates. It can be used to detect anomaly in any feature. It is not restricted to using a specific feature such as the mobility target feature for anomaly detection.

The key steps during the training process of BANBAD include analyzing raw datasets, categorizing each feature, computing beliefs of features from all samples, and finally extracting the belief range for the target feature. A normal profile is thus
generated. During the testing process, if the belief value of the target feature in a sample is found to be outside its acceptable range, then an anomaly is detected, otherwise, it is normal. To handle missing data in the testing sample, structure learning techniques such as PC [29] are used to generate a belief network which helps one to compute belief of the target feature using causal and diagnostic reasoning (evidences) from other features.

The development of BANBAD significantly contributes to the field of anomaly detection in a few ways. First, it describes a method of easily generating and maintaining a profile. It achieves both high detection rate ($\geq 95\%$) and low false alarm rate ($\leq 5\%$) for a chosen target feature, and false alarm rate can be bounded by certain predefined threshold. It also has the potential to function with an incomplete sample during the testing process. This function is useful in ad hoc networks, because its dynamically changing topology can result in the incomplete observations for the selected features. Moreover, BANBAD exhibits good performance for real datasets derived from different networks. Finally, the BANBAD toolkit allows one to use it for anomaly detection easily and effectively.

1.4 Dissertation Organization

The rest of the dissertation is organized as follows. In Chapter II we review related work. Chapter III describes mobility pattern based anomaly detection algorithm for mobile networks. Chapter IV proposes centralized BANBAD algorithm in detail as well as gives its overhead analysis. Chapter V presents the simulation
results using synthetic datasets. Chapter VI presents the experimental results using two real datasets. Chapter VII describes the software toolkit that implements BANBAD. Chapter VIII concludes the dissertation with a brief discussion and indications of future work.
In this chapter, we review some of the most relevant previous work in the intrusion prevention and detection areas of computers and network security.

The classifications of intrusion prevention and detection (IPDS) are given in Figure 2.1.

![Figure 2.1: Classifications of IDPS [66-67].](image)

An Intrusion Prevention System (IPS) is a device that monitors network and/or
system activities for malicious or unwanted behavior and can react, in real-time, to block or prevent those activities [66]; whereas an Intrusion detection system (IDS) is software and/or hardware designed to detect unwanted attempts at accessing, manipulating, and/or disabling computer systems, mainly through a network, such as the Internet [67]. Where IDS (passive security solution) informs of a potential attack, an IPS (active security solution) makes attempts to stop it.

From Figure 2.1, we observe that two main types are associated with IPS. A host-based IPS (HIPS) is where the intrusion-prevention application is resident on that specific IP address, usually on a single computer; where a network-based IPS (NIPS) is one where the IPS application/hardware and any actions taken to prevent an intrusion on a specific network host(s) is done from a host with another IP address on the network. Two main techniques for IPS are: encryption and authentication. In cryptography, encryption is the process of transforming information (referred to as plaintext) using an algorithm (called cipher) to make it unreadable to anyone except those possessing special knowledge, usually referred to as a key [68]. Authentication is the act of establishing or confirming something (or someone) as authentic, that is, that claims made by or about the subject are true. This might involve confirming the identity of a person, tracing the origins of an artifact, ensuring that a product is what its packaging and labeling claims to be, or assuring that a computer program is a trusted one [69].

From Figure 2.1, we observe also that four main types are associated with IDS. A network-based IDS (NIDS) is an independent platform which identifies
intrusions by examining network traffic and monitors multiple hosts; a host-based
IDS (HIDS) consists of an agent on a host which identifies intrusions by analyzing
system calls, application logs, file-system modifications (binaries, password files,
databases) and other host activities and state; a protocol-based IDS (PIDS) consists of
a system or agent that would typically sit at the front end of a server, monitoring and
analyzing the communication protocol between a connected device (a user/PC or
system) and the server; and an application protocol-based IDS (APIIDS) consists of a
system or agent that would typically sit within a group of servers, monitoring and
analyzing the communication on application specific protocols. Two main techniques
for IDS are: anomaly based and signature based. Anomaly based IDS establishes a
performance baseline based on normal network traffic evaluations. It will then sample
current network traffic activity to this baseline in order to detect whether or not it is
within baseline parameters. If the sampled traffic is outside baseline parameters an
alarm will be triggered; where in signature based IDS, network traffic is examined for
preconfigured and predetermined attack patterns known as signatures [67].

Some open source systems are shown on Figure 2.1, e.g., Snort [70] and
Untangle [71] are for both IPS and IDS, and OOSEC [72] is for IDS only.

Recall that IPS and IDS are two different solutions in that one is a passive
detection monitoring system and the other is an active prevention system. It would be
good that one evaluate a more mature IDS technology, and try the younger, less
established IPS solutions parallel. It is important to remember that no single security
device will stop all attacks all the time. IPS and IDS work best when integrated with
additional and existing security solutions.

2.1 Intrusion Prevention Techniques

Data encryption and authentication are two primary methods, and play an important role for intrusion prevention techniques. The basic idea behind such techniques relies on key management. Li et al. propose a static key management strategy, in which a key pre-distribution scheme is designed using the bivariate t-degree polynomial in a hexagonal coordinate system for the expected locations of the sensor nodes [7]. By comparing with the square-based polynomial pre-distribution scheme [9], the authors show that their scheme can improve the effectiveness of key management in terms of the probability of key establishment, and can extract appropriate security threshold with different polynomial degrees in sensor networks. In addition to static key management scheme, another type of key management scheme is the dynamic key management scheme in which keys can be updated periodically or on demand as a response to node capture. By performing key update, the compromised nodes are segregated and the network security can be enhanced. Li et al. propose a group-based dynamic key management scheme in wireless sensor networks without the requirement for a fixed infrastructure such as base stations and cluster heads [10]. Their scheme ensures the network security without tampering the compromised sensor nodes with an acceptable overhead, when \( k = l \), the overhead is minimum where \( k \) is the number of key polynomials known to each node and \( l \) is the number of polynomials unknown to each node.
Ma et al. propose an In Situ Pairwise Key (IPAK) bootstrapping algorithm for shared-key establishment between neighboring sensors [11]. Two sensor types, service sensors and worker sensors, are introduced. The simulation study shows their work can achieve high key-sharing probability with low storage in worker sensors. Ren et al. propose a location-aware multi-functional key management framework, which ensures both node-to-sink and node-to-node authentication along report forwarding routes, to guarantee end-to-end security in wireless sensor networks [12].

Recall that BANBAD is designed for detection and not prevention. We review intrusion detection techniques next.

### 2.2 Intrusion Detection Techniques

Intrusion detection technique serves as the second line of defense, and is an important component of the defense-in depth or layered network security mechanism. The two main intrusion detection techniques are misuse detection and anomaly detection. As to misuse detection techniques, Yang et al. propose a network misuse detection mechanism based on traffic log, combining the payload independent traffic classification technology [13]. Through observation and comparisons over extensive experiments, the authors complete the selection of behavior features, and by using collaborative learning method [14], they overcome the problems of both sample in sufficiency and adaptability.

For anomaly detection techniques, Zhang et al. present an anomaly detection technique in which each node locally analyzes available network data for anomalies.
Intrusion attempts are detected by employing a distributed cooperative mechanism in which all participating nodes cast votes according to data they have previously analyzed [3]. Results of this work are incomplete. First, trace data-feature or audit data source-design is not complete. It is not clear what information a routing protocol should include to make the IDS effective. Second, the detection model design does not indicate when to initiate intrusion response. Finally, their technique suffers from performance penalties and high false alarm rates.

Two main approaches exist in anomaly detection techniques: statistical-based and rule-based. For rule-based approach, Silva et al. define multiple rules by taking into account data messages in wireless sensor networks. These rules can be used to determine if a specific type of network failure has occurred and to raise an intrusion alarm if accumulative network failures exceed a predefined threshold [24]. Hilas presents a rule-based expert system which aims to detect superimposed fraud cases in the telecommunications network of a large organization [25]. The expert system incorporates the network administrator's knowledge along with observations and knowledge derived from the application of data mining techniques on historical data. The knowledge is expressed in the form of rules implemented by C4.5 [26] algorithm to classify calls into two classes, normal or anomaly.

For statistical-based approach, Sun et al. describe another technique by using (a) the high-order Markov model to specify the mobility pattern of a user; (b) EWMA (exponentially weighted moving average) for fading in order to maintain an updated profile of each user; and (c) use distance, a metric for indicating how closely a mobile
user follows her routines [4]. As they themselves address, the algorithm has high false alarm rates and low, dependent detection rates. Moreover, it is not easy to tell whether an anomaly exists when the speed ranges of nodes are fairly low. Chatzigiannakis et al. present a review and classification of data fusion algorithms [17], specifically addressing the anomaly detection problem. By comparing two different representative approaches, one based on the Dempster-Shafer Theory of Evidence [15], and the other based on Principal Component Analysis [16], under different attack scenarios, they identify which of these two approaches operates more efficiently, and could be used to detect a wide range of attacks in an integrated way. However, the crucial performance of anomaly detection, false alarm rate and detection rate are not exhibited in the paper. Liu et al. propose the insider attacker scheme [18]. By exploiting the spatial correlation among networking behaviors of sensor in close proximity, the scheme takes into consideration multiple attributes simultaneously without requiring prior knowledge about normal or malicious sensor activities.

Li et al. propose the group-based anomaly detection scheme for wireless sensor networks [19]. They use Mahalanobis distance measurement and the OGK estimators [20] in the intrusion detection algorithm to consider multiple attributes (features) of the sensor nodes to detect malicious network attack behaviors. By conducting real data [21] experiments and comparing with other intrusion detection schemes of [18], lower false alarm rate and higher detection rate are achieved. However, all the features are assumed to be normal distributed, and handling missing or incomplete data is not clearly addressed.
Alves et al. propose two anomaly detection methods based on the concept of profiles for detecting telecom fraud situations [22-23]. Some deficiencies are: first, it's not clear how to efficiently extract the threshold; second, they argue the profile should be always updated to avoid loss of information without considering the possibility of introducing error due to profile update; third, no false alarm rate and detection rate are demonstrated to evaluate the proposed methods.

Cai and Gupta propose an mobility-pattern-based anomaly detection algorithm for mobile networks [30]. Data mining and fuzzy logic techniques are used to generate a normal mobility profile during the training process, and to distinguish abnormal mobility patterns from normal ones. Good performance is achieved by efficiently tuning the threshold by trial and error. However, only one feature – mobility is considered in that paper.

In summary, for anomaly detection techniques, some work presents an IDS framework without implementation detail [3]. Some methods only achieve good performance when strong assumptions are met, like high velocity ranges [4-6]. Some work does not demonstrate the crucial performance in terms of both false alarm rate and detection rate [22-23, 25]. Some work [22-23] does not clearly specify how to extract the threshold – crucial design parameter to achieve the good performance. Some work considers only one target feature and limited datasets for testing [4, 19, 30]. In addition, most of the work does not clearly address how to handle the missing data during the testing process and adaptive learning techniques, and sometimes methods creating and updating dynamic profiles are very expensive [3]. Our work,
especially BANBAD, addresses all these deficiencies and gives the improvements.

2.3 Belief Networks

A belief network or directed acyclic graphical model is a probabilistic graphical model [55, 75] that represents a set of random variables and their conditional independencies via a directed acyclic graph (DAG). For example, a belief network could represent the probabilistic relationships between diseases and symptoms. Given symptoms, the network can be used to compute the probabilities of the presence of various diseases.

In belief networks, edges represent conditional dependencies; nodes which are not connected represent variables which are conditionally independent of each other. Efficient algorithms exist that perform inference and learning in belief networks.

Bayesian networks are used for modeling knowledge in computational biology and bioinformatics (gene regulatory networks, protein structure, gene expression analysis [56]), medicine [57], document classification, information retrieval [58], image processing, data fusion, decision support systems, engineering, gaming and law [59-61].

2.4 Structure Learning Techniques

Automatically learning the graph structure of a belief network is a challenge topic within machine learning. The problem of leaning network topology, however, is NP hard. We introduce two main structure learning techniques: K2 and PC.
K2 algorithm is a commonly used greedy search algorithm in belief structure learning. The performance of the K2 algorithm is greatly affected by the order of input nodes. If all parents in the node ordering occur prior to their children in the node ordering, the algorithm will perform optimally and consequently the results are very accurate [28]. The K2 algorithm is very efficient as the node-ordering information reduces the search space of DAG, thus making the search non-exhaustive. However, the performance of the algorithm may be poor when using wrong orderings in which most children nodes appear prior to their parents and for orderings that are random in nature. Unfortunately, in most cases, the input node ordering is usually unknown.

Another well-cited structure learning algorithm is PC algorithm [29] based on independence test. The basic hypothesis of PC algorithm is: the independence relationships have a perfect representation by DAG; we have a very large database; and statistical tests have no error. Under these conditions, the algorithm will discover and equivalent belief network. Computing p-value usually used to determine whether there is independence for two variables. However, sometimes the direction of links needs to be solved with user interaction to keep the DAG structure because PC algorithm can not necessarily finish complete arc orientation. Due to we have very large dataset, therefore, we apply PC algorithm mainly in our BANBAD work.
3.1 Model Description

3.1.1 Network model

As clusters and patterns are used a coordinate system can also be used to define our network model. The network is modeled as a generalized graph $G = (V, E)$. The vertex set $V$ represents the set of clusters. Edge set $E$ represents all the edges. If two clusters have a parent-child relationship, there exist several (at least one) edges (pattern strings) between the two.

3.1.2 Mobility model

We exclude the scenario that a node has totally random movement behavior and hence we do not consider the random walk model [62] in which the node will move to any one of the adjacent cells with equal probability after leaving a cell. In reality, random walk occurs very rarely, if at all. Further, one should capitalize on the most common and frequent occurrences. We assume that each node normally traverses with a destination in mind and the normal mobility profile of a node can be represented by a Multi-Leaf tree structure to capture the node movement. Each
possible route can be represented by a sequence of symbols $C_1, C_{11}, C_{514}...$ where $C$ stands for the cluster and the subscripts indicate the hops. For instance, $C_{584}$ refers to the cluster on the third tree level which passed from the fifth cluster of the first hop, via the eighth cluster of the second hop, to the destination cluster which is the fourth cluster of the third hop.

3.2 MPB Anomaly Detection Algorithm

3.2.1 Pattern definition

We use the combination of velocity and direction to represent the mobility pattern of nodes within any 1 hop by:

$$P = V \alpha$$

Where $V \in [0, V_{\max}]$, $\alpha \in [0, 2\pi]$, and $V_{\max}$ is the maximum possible velocity.

Therefore, any route can be represented as the sequence of $P$ elements depending on the hops. For instance, three hops route can be represented as $P_0P_1P_2$ or $(V_0\alpha_0)(V_1\alpha_1)(V_2\alpha_2)$.

3.2.2 Data mining and fuzzy logic approaches

In this subsection, data mining [63] and fuzzy logic [64] techniques we use for solving the challenging problem (See chapter I) of handling many closer points and look-alike mobility patterns are discussed.
Cluster Classification Algorithm:

Input: Destination points dataset $D$, # of representatives $K$.
Output: A set of clusters $C_1, \ldots, C_k$.

1: randomly select a point $p$ to form a new cluster;
2: for each new cluster $\eta$ do
   for each current cluster $C$ do
      calculate average distance $(C, p)$;
      assign $p$ to $\eta$ or $C$ based on distance measure;
3: update the profiles of newly formed clusters;
4: repeat Lines 1-3 until no new cluster are added;
5: return

Figure 3.1: Cluster classification algorithm.

Figure 3.1 is the pseudo code of the cluster classification algorithm, originated from data mining techniques, which is used in both training and testing processes in order to generate the corresponding clusters or detect anomaly by collecting certain points through the 'distance' measurement.

We use the following example to show how fuzzy logic techniques are applied in our MPB algorithm. By the normalized membership function $(1-x/n)$, one can ascertain how "close" the current velocity is to the designated velocity. Here, $x$ is the difference between current velocity and designated velocity, and $n$ is the predefined maximum velocity. Suppose 3.25 and 3.251 are the two current velocities.
and 4 is the designated velocity. We would then have \( (1-(4-3.251)/n) > (1-(4-3.25)/n) \) implying that 3.251 is closer to 4 than 3.25. Therefore, by applying fuzzy logic techniques, we can mine data smoothly during the training process and distinguish the subtle difference during the testing process.

3.2.3 The distance measure

We introduce two thresholds, \( C_{thr} \) and \( P_{thr} \), which are the design parameters for creating new clusters and pattern strings, respectively.

In the training process, we compare \( C_{thr} \) and/or \( P_{thr} \) with the distance among the points calculated using data mining and fuzzy logic techniques and decide to add new clusters and/or new pattern strings for an existing cluster.

In the testing process, still based on the calculated distance, we decide whether there exists an anomaly. More specifically, when distance \( (C_x) \leq C_{thr} \) and distance \( (P_y) \leq P_{thr} \), the route with destination point \( P \) is evaluated as normal, otherwise the route is identified as an anomaly.

3.2.4 Multi-leaf tree structure

We use Multi-Leaf tree structure to model the normal mobility profile of nodes, consider the example of Figure 3.2 which depicts a typical mobility profile associated with a specific node.
Figure 3.2: Multi-leaf tree structure.

Such a Multi-Leaf tree structure is generated by data mining techniques for creating clusters and fuzzy logic techniques for creating the corresponding set of pattern strings of each cluster.

- The root, starting cluster, is composed of a group of points that are within $C_{thr}$ distance of each other, where $C_{thr}$ is the threshold of cluster;
- The first-level nodes, such as $C_1$, $C_2$, and $C_m$ are the distinct destination cluster nodes which we’ve collected after certain time periods;
• $C_{11}$ and $C_{1n}$ are the two possible children cluster nodes whose parent cluster node is $C_1$, that is, starting from $C_1$, after certain time periods, there exists two possible subsequent distinct destination cluster nodes;

• $C_{1n}$ is generated \(\text{iff} (\forall x \in [1, n-1]), \exists (C_{1x} \leq C_{thr})\) where $C_{1x}$ is the cluster distance of point $P$; similarly, the $n^{th}$ pattern string associated with $C_{1n}$ is generated \(\text{iff} (\forall y \in [1, n-1]), \exists (P_y \leq P_{thr})\) where $P_y$ is the pattern string distance of point $P$ and $P_{thr}$ is the threshold of pattern strings.

• The rectangle box attached to each cluster specifies the corresponding set of pattern strings. For each pattern string, we have the information of both the pattern distributions and pattern orders;

• This process is continued until we get to the final level destination clusters which are the leaf nodes of Multi-Leaf structure. Note that all of the above information can be gathered by data mining and fuzzy logic techniques using algorithms similar to the one of Fig.2.

3.2.5 The framework

The general framework of our proposed MPB anomaly detection algorithm is given in Figure 3.3.
Figure 3.3: General framework of MPB anomaly detection algorithm.

The purpose for the training process is to generate and maintain an up-to-date normal mobility profile for any specific node, and by comparing the distance metric of the testing data with that of the current profile, we can distinguish whether testing data is an anomaly.

3.2.6 Overhead of MPB anomaly detection algorithm

Let us consider the following general case. Note that we assume all training and testing data are collected using the same hops starting from the same starting cluster.
Suppose we have a partial 2-level Multi-Leaf tree structure composed of the starting root cluster and \( k \) first level destination clusters. There are \( n \) hops from the root to the first level destination clusters. For each of the first level destination clusters, there are \( m \) pattern strings specifying every possible route from the root. Here, \( m \) is a variable since it's possible for \( m \) to be different for different first level destination clusters. Let \( N \) be the total number of points for constructing such partial 2-level Multi-Leaf tree structure.

Let \( P(x, y) \) be a point with \( P_1P_2...P_n \) as the pattern string. We integrate \( P \) into the Multi-Leaf tree structure as follows.

**For training process:** In the first step, we need to determine distances from \( P \) to every other point resulting in \( N \) distance calculations; In the second step, for each cluster, average cluster distance from \( P \) needs to be computed. Therefore, we have \( k \) average distance calculations; In the third step, we need to get the minimum average cluster distance (MinACD); In the fourth step, we need to compare MinACD with \( C_{thr} \) and determine whether a new cluster needs to be introduced.

1. If (MinACD > \( C_{thr} \)), add \((k+1)^{st}\) cluster associated with its mobility pattern string \( P_1P_2...P_n \). **DONE.**

2. Else, select the cluster which has the MinACD, say, \( C \).

2.1) we compute a total of \( m \) distances of \( P \) from the \( m \) pattern strings of \( C \);

2.2) we compute \( n \) average pattern strings distance of \( P \) from the \( n \) hops of \( C \);

2.3) determine the minimum average pattern strings distance (MinAPSD);

2.4) if (MinAPSD > \( P_{thr} \)), add \((m+1)^{st}\) pattern string, that is, \( P_1P_2...P_n \) to \( C \). **DONE.**
It's now clear that if the MPB algorithm stops in step 1, then the overhead is $O(N + 2k)$ whereas if the algorithm quits in step 2.4, the overhead is $O(N + 2k + 3m)$.

For testing process: we just need to decide whether point $P$ is an anomaly. Same steps are used in the training process, the only difference being instead of DONE in step 1 and 2.4, we generate an ALERT.

Therefore, the overall overhead of the MPB algorithm is $O(N + 2k + 3m)$.

3.3 Simulation Results

Without loss in generality, we generate a typical normal mobility profile which accounts for the regular mobility patterns of a specific node. Such data sets are obtained using statistical methods. Ten thousand records are generated with random velocity and degree, where $velocity \in [1, 8]$ and $degree \in [0, 2\pi]$. All values are of type double.
A Multi-Leaf tree structure with three hops is acquired by data mining techniques, more specifically, cluster classification algorithm in Figure 3.1. After this process, we have eight clusters as the first level destination clusters, seven clusters as the second level destination clusters and just one - $C_{111}$ as the third level destination cluster. A specific route ($C_1 \rightarrow C_{11} \rightarrow C_{111}$) connected with the corresponding points, i.e., $P_1 (2.17, 4.92) \rightarrow P_2 (4.84, 1.64) \rightarrow P_3 (7.93, 1.28)$ is shown in Figure 3.4.

Note that the mobility profile we generated is general and typical for most nodes. It may not be suitable for nodes with totally random movement behaviors, but recall that totally random model is out of the scope of this dissertation.
False alarm rate and detection rate are selected as the performance metrics for our proposed algorithm [7-9].

1. $C_{thr} = 5.0$ and $P_{thr} = 0.6$: The false alarm rate and detection rate are plotted in Figure 3.5.

![Figure 3.5: Performance at different velocity ranges.](image)

Figure 3.5 demonstrates that the detection rate is fairly high and the false alarm rate is fairly low. Unlike the results of Sun et al., whose performance was dependent on the velocity ranges, this MPB algorithm achieves good detection rate (90%) and false alarm rate (8%) for all ranges of velocity.

Good performance is achieved when $P_{thr} = 0.6$. Given, say $C_{thr} = 5.0$, naturally the question of whether $P_{thr}$ can be determined arises. Since velocity 3 and 4 are crucial for the performance (from Figure 3.5, they achieve worst performance for both
the detection rate and the false alarm rate), we use these 2 velocity ranges for further
testing.

![Fixed Pthn](image)

**Figure 3.6:** Fixed mobility pattern threshold.

2. $P_{thr}$ is decidable: From Figure 3.6, we observe that $P_{thr}$ can be efficiently
determined. The detection rate is increased during the range of $P_{thr}$ from 0.2 to 0.7,
but the corresponding false alarm rate is 0% when $P_{thr} \leq 0.4$, theoretically, it's
impossible. And although detection rate becomes better when $P_{thr} = 0.7$, the
corresponding false alarm rate also increases, to the rate of almost 30%, this is also
not acceptable. Therefore, selecting the value of $P_{thr}$ around 0.6 might be the ideal
value for performance in terms of both the false alarm rate and the detection rate
given certain $C_{thr}$ (here, $C_{thr} = 5.0$). After testing for $P_{thr} = 0.59$, 0.6 and 0.61,
respectively, our selection has been confirmed, that is, once $P_{thr}$ is around 0.6, the
good performance of both detection rate and false alarm rate is achieved. Thus, we can say that value of $P_{\text{thr}}$ can be efficiently determined for a given $C_{\text{thr}}$, note that the whole process to determine $P_{\text{thr}}$ is adjusted by trial and error.
CHAPTER IV

CENTRALIZED BANBAD ANOMALY DETECTION ALGORITHM

The basic idea of BANBAD during the training process includes analyzing raw datasets, categorizing each feature, computing beliefs of features from all samples, and finally extracting the belief range for the target feature. A normal profile is thus generated. During the testing process, if the belief value of the target feature in a sample is found to be outside its acceptable range, then an anomaly is detected, otherwise, it is normal. To handle missing data in the testing sample, structure learning techniques such as PC [29] are used to generate a belief network which helps one to compute belief of the target feature using causal and diagnostic reasoning (evidences) from other features.

4.1 Assumptions

First, we assume there is a secure station in the ad hoc network which maintains normal profiles of nodes by executing the training process. The secure station is also in charge of detecting anomaly by executing the testing process. Assuming an existence of a secure station to monitor security policies is reasonable and justified because one has to place or build trust in a node (or set of nodes). Furthermore, our focus in this dissertation is to show the viability, power and
efficiency of anomaly detection technique using Belief Networks (BNs) and Bayesian statistics. Naturally, BANBAD can be extended to a distributed case, however, that is out of the scope of this dissertation and left as a future work. The centralized approach simplifies the discussion without compromising its usefulness. The communication between nodes and secure station is considered reliable.

Second, we assume each node has a specific behavior database which describes its normal activities. All node behavior databases are stored in a secure place which is hard to be compromised. There exists at least one device inside each mobile node which can provide the accurate behavior of the node at any time. Even if an attacker uses the captured node, he still does not know how to manipulate such device which is inside the node.

Third, we assume nodes have fairly regular behaviors. Therefore, it is viable to create the normal profile for each node. We assume normal profile generated by perfect data which follows normal routine and exclude totally random behavior by a node. Indeed, based on belief networks and the probability distributions we described in next section, such normal profile can be generated under any appropriate scenario as long as normal behavior exists during a certain time period, and this is not limited to ad hoc networks. Therefore, BANBAD is also applicable to other networks.

Finally, our current focus is to simply show the applicability of BANBAD in anomaly detection with bounded false alarm rate under very relaxed assumptions and not to limit its applicability to any specific type of networks. Hence, our discussion does not entail exploitation of various specific features of ad hoc networks. Moreover,
the features selection process is out of the scope of this dissertation; our examples illustrate few viable features and are by no means exhaustive. However, based on certain techniques such as structure learning [28-29], we can test if the selected features are associated or not.

4.2 Belief Networks and BANBAD Preliminaries

Recall that a Belief Network (BN) is a directed acyclic graph (DAG) in which nodes represent variables, or features; and arcs represent the nature of dependence among the features and the conditional probabilities.

In a hypothetical arc from feature $A$ to feature $B$, $A$ is the parent of $B$. $B$ depends directly on $A$. All other local probability distributions are conditional. If a feature is observable, the feature is an evidence feature.

Now, consider a BN that is used to detect anomalies in an ad hoc network. Suppose energy consumption of a node is related to average velocity, displacement, and some other features. The dependencies among these features can be represented, in a simplified scenario, for example, by a chain model as in Figure 4.1.

![Figure 4.1: The chain application model displaying the dependencies of various features of a node.](image)
As Figure 4.1 shows, energy consumption is affected by Displacement ($D$) and it affects Response Time ($R$). Note that in this example, for brevity, we do not display all factors that may affect energy consumption ($E$). The continuous raw dataset in a BN is categorized based on some characteristic features. For example, Average Velocity ($V$) may be partitioned into two states \{$V_1, V_2$\} where $V_1 = \left[ v_{min}, v_1 \right]$ and $V_2 = \left[ v_1, v_{max} \right]$. Similarly, there are two states \{$D_1, D_2$\} for Displacement ($D$), where $D_1 = \left[ d_{min}, d_1 \right]$ and $D_2 = \left[ d_1, d_{max} \right]$; three states \{$E_1, E_2, E_3$\} for Energy Consumption ($E$), where $E_1 = \left[ e_{min}, e_1 \right]$, $E_2 = \left[ e_1, e_2 \right]$, and $E_3 = \left[ e_2, e_{max} \right]$; and three states \{$R_1, R_2, R_3$\} for Response Time ($R$), where $R_1 = \left[ r_{min}, r_1 \right]$, $R_2 = \left[ r_1, r_2 \right]$, and $R_3 = \left[ r_2, r_{max} \right]$.

To illustrate BANBAD, we use energy consumption ($E$) with 3 states as our target feature. We detect anomalies, say, for the first state $E_1$ of $E$ as:

**Case 1: A complete testing sample for $E$ exists.** We can compute the probabilities of the three states during the training process. For example, during a certain time period, we observe the probability vector \{0.7, 0.1, 0.2\} where 0.7, 0.1 and 0.2 are the probabilities of $E_1$, $E_2$, and $E_3$, respectively. Here, 0.7 is the causal reasoning [27] of $E_1$ (probability inferred from the parent node, $D$, that $E_1$ occurs), referred to as $\pi(E_1)$, i.e., there is 70% chance that $E_1$ occurs in the training dataset available to us. Let us now assume that we obtain two probability vectors \{0.5, 0.2, 0.3\} and \{0.65, 0.21, 0.14\} for $E$ within the same time period from two different
testing data. The next step, naturally, is to evaluate the differences between the training data and the two testing data.

The Belief ($Bel$) vector for $E$ can now be computed using Bayes’ rule\(^1\). It is a good metric for anomaly detection since it is assigned when the relevant evidence is taken into account. It combines causal reasoning and diagnostic reasoning, where diagnostic reasoning of $E_i$ is the probability inferred from the child node, $R$, that $E_i$ occurs, referred to as $\lambda(E_i)$. Here, $R$ is a childless node, and initially we arbitrarily set $\lambda(R)=1.0$.

Continuing with the numerical example, we can compute the diagnostic reasoning of $E_i$ given $R$ in the training data to be 1 by applying $\lambda(E_i) = \sum R \lambda(R) \cdot P(E_i | R)$, i.e., $\lambda(E_i_{\text{training}})=1.0$, $P(E_i | R)$ is the probability that $E_i$ occurs given $R$. Then, we obtain $Bel(E_i_{\text{training}})=0.7$. Since we have complete datasets for $E$ and to account for several training samples, we can use

$$\lambda(E_i) = \pi(E_i) / \bar{\pi}(E_i_{\text{training}})$$  \hspace{1cm} (1)$$

$E_i$ refers to individual training or testing sample and $\bar{\pi}(E_i_{\text{training}})$ refers to the weighted mean of all causal reasoning values of $E_i_{\text{training}}$. Thus, the diagnostic reasoning of the two testing data are $\lambda(E_i_{\text{testing1}}) = \frac{0.5}{0.7}$ and $\lambda(E_i_{\text{testing2}}) = \frac{0.65}{0.7}$, respectively. So, $Bel(E_i_{\text{testing1}}) = \frac{\pi(E_i_{\text{testing1}}) \times \lambda(E_i_{\text{testing1}})}{\sum \pi(E_i_{\text{testingi}}) \times \lambda(E_i_{\text{testingi}})} \approx 0.30$ and $Bel(E_i_{\text{testing2}}) \approx 0.53$. 


This implies that the first state, $E_1$, of the second testing data {0.65, 0.21, 0.14} is closer to the training data than the first testing data. With the design parameter threshold $r$ properly set, by comparing the difference of the belief between the training and the testing data, we can distinguish whether the specific testing data is an anomaly or not. Difference is denoted as distance and is discussed in detail in subsection C. This case is relatively straightforward, since anomalies can be detected easily given the existence of complete testing samples.

**Case 2:** A complete testing sample for $E$ does not exist. The sample is incomplete due to some missing information. A modification of BN techniques allows us to detect anomalies.

To further explore how this is possible, first consider the belief propagation algorithm, one of the exact Bayesian inference algorithms. Energy consumption ($E$) of Figure 4.1 is still the target feature. Based on the belief propagation algorithm:

1. $\text{Bel}(E) = a \pi(E) \cdot \lambda(E)$ — Belief of energy consumption

2. $\pi(E) = \pi(D) \circ M(E | D)$

3. $\lambda(E) = M(R | E) \circ \lambda(R)$

Where

- $\cdot$ — term by term product of two vectors;
- $\circ$ — dot product of two vectors;
- $a$ — normalizing constant;

The belief propagation algorithm is used to update the belief of $E$ given

$$Bel(X_i) = \frac{\pi(X_i) \times \lambda(X_i)}{\sum_i \pi(X_i) \times \lambda(X_i)}$$
evidence from its related features. Once executed, its parent Displacement \((D)\) transmits the causal reasoning \((\pi \text{ message})\) and its child Response Time \((R)\) transmits the diagnostic reasoning \((\lambda \text{ message})\) to \(E\) for computing the belief of \(E\).

Continuing with the example, the conditional probability distribution \((CPD)\) of \(E\) given \(D\) and \(CPD\) of \(R\) given \(E\) (equations (2) and (3)) allow the derivation of the causal reasoning (4) and diagnostic reasoning (5) of the target feature \((E)\).

\[
M(E \mid D) = \begin{bmatrix} 0.5 & 0.4 & 0.1 \\ 0.1 & 0.3 & 0.6 \end{bmatrix} 
\]

\[
M(R \mid E) = \begin{bmatrix} 0.2 & 0.4 & 0.4 \\ 0.3 & 0.3 & 0.4 \\ 0.6 & 0.1 & 0.3 \end{bmatrix} 
\]

\[
\pi(E) = 0.5\pi(D_1) + 0.1\pi(D_2) 
\]

\[
\lambda(E) = 0.2\lambda(R_1) + 0.4\lambda(R_2) + 0.4\lambda(R_3) 
\]

\(\pi(E)\) depends directly on the causal reasoning of \(D\), and \(\lambda(E)\) depends directly on the diagnostic reasoning of \(R\). Now, detection can proceed as in case 1, i.e., belief can be computed to ascertain anomalies.

BNs allow us to learn causal relationship among features and handle incomplete datasets easily [27]. Hence, with indirect belief computation which relies on the evidence from other features, we can still detect anomaly.

4.3 The Directed Acyclic Graph (DAG) Model

Figure 4.1 uses the chain model, a special case of the DAG model, to explain how BNs are used for anomaly detection. A realistic and powerful DAG incorporates
more features. This is necessary because energy consumption of a node, for example, can be affected not only by its displacement but also by its local computation and communication. Such a DAG model is displayed in Figure 4.2.

Figure 4.2: A more complete DAG application model representing the various factors that affect energy consumption of a node.

### 4.4 Training and Testing Processes

The training process of BANBAD as depicted in Figure 4.3 first collects raw data and features. Then, it applies structure learning techniques [28-29] to generate a profile. Feedback (shown using dashed line) can be used to dynamically update the profile. Note that the DAG structure may not be unique due to existence of multiple samples. For simplicity, in this dissertation, we only consider a unique structure generated by combining all the training samples. Using this process, it is easy to maintain the profile over any time period.
The testing process (shown in Figure 4.4) consists of data and feature collection and evidence extraction of features. If there is an incomplete data for a target feature, BANBAD applies the belief propagation algorithm. By using corresponding profile—generated during the training process within the same time period, BANBAD computes the difference of the belief between the training and testing data. A significant deviation (≥ threshold $\tau$ ) indicates an anomaly. The difference is denoted as distance and is defined as

$$\text{Distance}(S) = |\text{Bel}_v(S) - \text{Bel}_e(S)|$$  \hspace{1cm} (6)

$S$ is a state of the target feature, $\text{Bel}_v(S)$ is the belief of $S$ in the training data, and $\text{Bel}_e(S)$ is the belief of $S$ in the testing data. If $\text{Distance}(S) \leq \tau$, $S$ is considered normal, otherwise $S$ is considered an anomaly.
4.5 Range Settings

Each state of a feature has its own true (normal occurrence) and false (does not occur) ranges, for example for $D_1$,

$$\left[0, \pi(D_1) - \varepsilon\right] \cup \left[\pi(D_1) - \varepsilon, \pi(D_1) + \varepsilon\right] \cup \left(\pi(D_1) + \varepsilon, 1\right]$$ (7)

For some $\varepsilon$, $0 < \varepsilon < 1$, $\pi(D_1)$ is the weighted mean of the causal reasoning of the first state $D_1$ of feature $D$ when multiple training samples exist. $\pi(D_1) - \varepsilon$ is the lower bound of the true range of $D_1$ and $\pi(D_1) + \varepsilon$ is the upper bound of the true range of $D_1$.

From Figure 4.1, we observe that $\pi(E)$ of the energy consumption depends directly on the causal reasoning of $D$, and $\lambda(E)$ depends directly on the diagnostic
reasoning of \(R\). Using (1), the true range of the diagnostic reasoning of the first state \(R_1\) of feature \(R\) is:

\[
\left[ 1 - \frac{\varepsilon}{\pi(R_1)}, 1 + \frac{\varepsilon}{\pi(R_1)} \right]
\]

and the false ranges of \(R_1\) are:

\[
\left[ 0, 1 - \frac{\varepsilon}{\pi(R_1)} \right] \text{ and } \left( 1 + \frac{\varepsilon}{\pi(R_1)}, +\infty \right)
\]

The true and false range settings above allow us to observe which evidences we have at the beginning of the testing process.

### 4.6 Belief Computation

Recall that we compute \(\text{Distance}(S) = |\text{Bel}_r(S) - \text{Bel}_c(S)|\) and compare it with the threshold value, \(\tau\), for anomaly detection. Thus, the crucial steps are the belief computation from both the training and the testing datasets. Given the true and false ranges of parent and children features, for brevity, let us discuss these steps below using a chain model as in Figure 4.1. We consider only 3 features, namely Displacement \((D)\), Energy Consumption \((E)\) and Response Time \((R)\). We now assume each feature has 4 states for clarity (rather than 2 states for \(D\), 3 states for \(E\) and \(R\)).

**Computing belief during the training process**: We again assume that feature \(E\) is the target feature for anomaly detection. Belief computation of feature \(E\) during the training process is shown in Figure 4.5. There are 3 steps.
Figure 4.5: Belief computation during the training process. $\pi$, $\text{Bel}$, $\lambda$ are ranges, e.g., $\pi(x):[\pi_w, \pi_u]$ where $\pi_w$ ($\pi_u$) is the lower bound (upper bound) of the causal reasoning of $x_i$.

**Step 1: Causal reasoning ($\pi$) computation:** Given complete raw dataset, the causal reasoning of each state of all the features can be observed. Let us say the causal reasoning of state $X_i$ is $\pi_{x_i}$, for $1 \leq i \leq 4$, and $X=D, E$ or $R$.

Individual training data just represents one sample and hence after constant learning with multiple samples, a range around $\pi_{x_i}$ can be defined to account for normal occurrence. This results in associating the range $[\pi_w, \pi_u]$ to the normal occurrence of $x_i$. Let $N =$ total size of $n$ samples. By applying the weighted rule,

$$
\pi_{x_i} = \sum_{j=1}^{n} \frac{\text{size of sample } j}{N}
$$

(10)

we adapt our predefined range accordingly (i.e., weighted mean if sample sizes are different). Hence, for each additional sample (say $n^{th}$) of training, the
updated causal reasoning is simply

\[
\frac{n^{th}\ \text{sample size}}{N} \pi(n^{th}) + \frac{\text{size of } (n-1) \text{ samples}}{N} \pi(n-1)
\] (11)

All the conditional probability distributions (CPDs) are computed as well, for example, the updated \( P(E \mid D) \) is

\[
\frac{n^{th}\ \text{sample size}}{N} P(E \mid D)(n^{th}) + \frac{\text{size of } (n-1) \text{ samples}}{N} P(E \mid D)(n-1)
\] (12)

and used in the testing process. Note that CPDs are fairly stable since we exclude totally random behavior. This way, we compute the entries of the \( \pi \) column in Figure 4.5.

**Step 2: Diagnostic reasoning (\( \lambda \)) computation:** To get the diagnostic range of each feature, we map 1-1 from \( \pi \) to \( \lambda \) as \( \lambda_\pi = \frac{\pi(x)}{\pi(x)} \) for each individual sample, \( \bar{\pi}_x \) is the weighted mean of all \( \pi \) values of \( x_i \). The entries of the \( \lambda \) column in Figure 4.5 are thus completed.

**Step 3: Belief computation:** To get the belief of the target feature \( E \) and/or all the features, we apply Bayes’ rule as mentioned in subsection A. Multiple samples lead the belief range to \([\text{Bet}_E^\alpha, \text{Bet}_E^\beta]\) as shown in the belief column of Figure 4.5.

**Computing belief during the testing process:** When we have incomplete testing sample, Bayesian inference algorithms need to be incorporated. This is shown in Figure 4.6 that depicts the BANBAD testing process with missing data. The basic idea is to update the range of the causal and diagnostic reasoning of the target feature.
in a number of iterations. Each iteration partially fills the gap created from the
missing data.

1: iteration = 1;
2: anomaly = false;
3: Compute the ranges $[\pi_{d1}^-, \pi_{d1}^+], [\pi_{e1}^-, \pi_{e1}^+], \text{and} [\lambda_{r1}^-, \lambda_{r1}^+]$;
4: while (iteration < maxIterations && anomaly == false) {
5: Bayesian inference with $\eta$ message passing from $D$ to target feature $E$;
6: if ($\exists i [\pi_{ei}^-, \pi_{ei}^+] \cap [\lambda_{ri}^-, \lambda_{ri}^+] \neq \phi$) {
7: anomaly = true;
8: break; }
9: else {
10: Update ranges of $\pi(E)$;
11: Compute $\lambda(E)$ from $\pi(E)$;
}  
12: Bayesian inference with $\eta$ message passing from target feature $E$ to $R$;
13: if ($[\pi_{ri}^-, \pi_{ri}^+] \cap [\lambda_{ri}^-, \lambda_{ri}^+] \neq \phi$) {
14: Update ranges of $\lambda(R_i)$;
15: Bayesian inference with $\lambda$ message passing from $R$ to target feature $E$;
16: if ($\exists i [\lambda_{ei}^-, \lambda_{ei}^+] \cap [\lambda_{ri}^-, \lambda_{ri}^+] \neq \phi$) {
17: anomaly = true;
18: break; }
19: else {
20: Update ranges of $\lambda(E)$;
21: Compute $\pi(E)$ from $\lambda(E)$;
}  
22: Bayesian inference with $\lambda$ message passing from target feature $E$ to $D$;
23: if ($[\pi_{di}^-, \pi_{di}^+] \cap [\lambda_{di}^-, \lambda_{di}^+] \neq \phi$) {
24: Update ranges of $\lambda(D_i)$;
25: iteration++;
}  
26: if (anomaly == true) generate alert;
27: else Compute belief of target feature $E$;

Figure 4.6: Testing process of BANBAD for the chain $D \rightarrow E \rightarrow R$. 
The policy for us to detect an anomaly is based on the target feature as a whole instead of its individual states. Suppose the target feature has m states. Using the training data, we compute the belief probabilities of these m states. In order to bound the false alarm rate to 5%, the lower bound of each interval is set to the \((5/m)\)th percentile and the upper bound is \((100-5/m)\)th percentile. Then, for a testing sample, if the computed probabilities of all the states fall into their own normal (belief) range, we conclude that there is no evidence of anomaly; otherwise, an anomaly is detected.

**Overhead analysis of the chain model:** Let \(r\) be the sample size observed in a day, \(d\) be the number of days in the training data (replicates), \(f\) be the number of features, and \(m\) be the maximum number of states in any feature. Note that parent and children may not have the same number of states. The overhead to compute \(\pi\) is \(O(r)\).

The worst case overhead in the remaining steps of the BANBAD training process when chain model is used can be shown as:

\[
\begin{align*}
&\quad \frac{O(m)}{Update \pi \ \text{Step 1}} + \frac{O(m^2)}{Update \ C PDs \ \text{Step 2}} + \frac{O(md)}{Step 3} + \frac{O(md)}{Step 4} \\
&= \text{(13)}
\end{align*}
\]

The worst case overhead of the testing process when chain model is used can similarly be derived as:

\[
\begin{align*}
&\quad \left( \frac{O(m)}{\pi \ & \lambda \ \text{Computation \ inference \ \text{range \ update}} \ \text{iteration}} + \frac{O(m^2)}{\text{maxIterations}} + \frac{O(m)}{\text{maxIterations}} \right) \\
&= \text{(14)}
\end{align*}
\]

In addition to the computation overhead, the worst case overhead of the communication in the training and testing process are \(O(rfd)\) and \(O(rf)\), respectively.
CHAPTER V

SIMULATION RESULTS OF CENTRALIZED BANBAD

BANBAD has been extensively tested under different scenarios. We performed experiments that include testing both synthetic and real datasets. We discuss these experimental results in this chapter and the next. As claimed earlier, BANBAD’s performance makes it a great candidate for anomaly detection. We conducted 35 experiments under different scenarios to test the correctness, and robustness of the centralized BANBAD.

5.1 Synthetic Datasets

For demonstrative purpose, we consider three scenarios. Scenario 1, the features follow normal distribution with linear relationship among them; Scenario 2, the features follow gamma distribution with linear relationship among them; and Scenario 3, the features follow normal distribution with non-linear relationship among them.

5.1.1 Normal distribution

We test our BANBAD technique using three different simulated datasets.
These datasets consist of six features for 100 days (replicates). The dimension of each dataset is \( r \times 6 \times 100 \), recall that \( r \) is the sample size observed in a day. We choose three different values for \( r \), namely 100, 1000, and 10000. Without loss in generality, we assume that the first feature, \( f_1 \) follows normal distribution with mean, \( \mu = 30 \) and variance, \( \sigma^2 = (20/3)^2 \); other features \( f_2 \) to \( f_6 \) are generated using the following scheme (we intend to have some linear relations among the features), Scheme 1:

\[
\begin{align*}
    f_2 & : 1/2 \times f_1 + \delta_1 \\
    f_3 & : 2 \times f_2 + \delta_2 \\
    f_4 & : 3 \times f_1 + \delta_3 \\
    f_5 & : 1/2 \times f_4 + \delta_4 \\
    f_6 & : 1/3 \times f_5 + \delta_5, \\
\end{align*}
\]

where each \( \delta_i \) is independent and identically distributed (iid) standard normal with mean 0 and variance 1.

After raw data generation, we categorize each feature using the scheme described in Table 5.1.

**Table 5.1: Categories of 6 features for Scheme 1**

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>( f_1 )</td>
<td>( f_2 )</td>
<td>( f_3 )</td>
<td>( f_4 )</td>
<td>( f_5 )</td>
<td>( f_6 )</td>
</tr>
<tr>
<td>&lt;30</td>
<td>&lt;15</td>
<td>&lt;25</td>
<td>&lt;55</td>
<td>&lt;45</td>
<td>&lt;18</td>
</tr>
<tr>
<td>&gt;=30</td>
<td>&gt;=15</td>
<td>[25, 35)</td>
<td>[55, 81)</td>
<td>[45, 65)</td>
<td>&gt;=18</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&gt;=35</td>
<td>[81, 93)</td>
<td>&gt;=65</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>[93, 99)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>[99, 115)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>[115, 130)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>&gt;=130</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
From Table 5.1, we see that \( f_1, f_2, \) and \( f_6 \) have two states; \( f_3 \) and \( f_5 \) have three states; and \( f_4 \) has seven states. PC structure learning technique [29] is applied to six categorical features obtained using the scheme described in Table 5.1. Figure 5.1 is the profile structure learnt by PC. As we can see, \( f_6 \) is the root which has no parent node, \( f_1 \) is the leaf which has no child node. Feature \( f_3 \) directly affects \( f_1 \) and \( f_2 \), and \( f_3 \) is affected by \( f_4, f_5, \) and \( f_6 \) directly.

![Figure 5.1: Structure of the profile learnt by PC.](image)

To assess the performance, we conduct a detailed study of false alarm rate (FAR) and detection rate (DR) for our proposed anomaly detection algorithm.

To evaluate FAR, three separate testing datasets \( D_1, D_2, \) and \( D_3 \) with dimension 1000, 2000 and 5000, respectively are generated for 100 replicates. \((D_i)^j\) is denoted as the \( j^{th} \)(\(1 \leq i \leq 3, 1 \leq j \leq 100\) replicate of dataset \( D_i \). We keep the same distributional scheme as in the training data described earlier. On the other hand, to
evaluate the DR, three different testing datasets D4, D5, and D6 each with dimension 1000, 2000, and 5000 are generated for 100 replicates. \((Di)_j\) again denotes as the \(j^{th}\) \((4 \leq i \leq 6, 1 \leq j \leq 100)\) replicate of dataset \(Di\). We intentionally use a different distribution for \(\beta\) for each dataset, to make it different from the training data. For other features, we keep the same linearly dependent structure as described in Scheme 1. Comparing the normal profile (i.e., the training dataset) with each of the replicates \((Di, 1 \leq i \leq 3)\) separately, when the estimated state probabilities of the replicate falls outside the normal range, then it is a false alarmed scenario. Hence, if \(\alpha\) replicates out of 100 have a belief probability outside the normal range, then the false alarm rate is \(\frac{\alpha}{100} \times 100\%\); similarly, comparing the normal profile with replicates of dataset \((Di, 4 \leq i \leq 6)\) separately, if the belief value falls inside the normal range, then miss-detection rate is \(\beta\%\) leading to a detection rate of \((100 - \beta)\%\). Note that during this evaluation process, single replicate is to be compared with the training data at a time.

The FAR and DR are plotted in Figure 5.2 for all features for the training dataset of dimension 1000 and the evaluation dataset of dimension 5000.
Recall the decision rule addressed in Chapter IV, our anomaly detection technique is based on the feature as a whole rather than its individual states. Based on our testing environment settings mentioned before, the dataset of $\mu = 30$ is used for testing the FAR. From Figure 5.2, we observe the FAR of all features to be almost 0% which indicates all belief values of all states fall into their own normal ranges. The datasets with mean $\mu = 18, 27, 33, 42$ are used for testing the DR. From Figure 5.2, we observe that DR of all the features is almost 100% which indicates at least 1 belief value of any state falls outside of its normal range.
Results of Figure 5.2 validate our BANBAD technique for anomaly detection using all the features in our decision. Now let us assume feature 4 to be the target feature. We explore the FAR and DR in more detail for individual states of feature 4 which are plotted in Figure 5.3 and 5.4. The training dataset used is of 1000 dimension; similar results were obtained when normal profile of other dimensions were used.

![Graph showing FAR for different dimensions](image)

Figure 5.3: False alarm rate of feature 4.

We set the maximum value of probability of Type-I error at 2%. In statistics, the term Type I error, also called $\alpha$ error or false positive is used to describe possible errors made in a statistical decision process [76]. Plainly speaking, it occurs when we
are observing a difference when in truth there is none. This is false alarm rate in our case. Feature 4 has 7 states, then the lower bound of a 98% belief range is 1st percentile and the upper bound is 99th percentile. From Figure 5.3, we observe that the FAR of all the states of feature 4 for dataset D1 and D2 is between 0 and 2%. For dataset D3, it is very close to 0%. Clearly, this implies that BANBAD performs as expected, i.e., one can bound the FAR to a predefined percentage and it is stable.

![Detection rate of feature 4](image.png)

Figure 5.4: Detection rate of feature 4.

Figure 5.4 shows the detection rate for various states of feature 4 using $\mu = 42$ and training dataset of dimension 1000 with predefined 5% bound on FAR. Recall that D4, D5, and D6 with dimension 1000, 2000, and 5000 each were designed to be anomaly. In fact, the DR is at least 95%. We can see an excellent performance
for most states except state 2 which appears to show somewhat of an erratic behavior. This erratic behavior can be explained due to the way categorization of raw data in the training dataset occurs. The categorization affects the probability distribution of a feature and DR depends on this underlying distribution. In practice, one has little control over the categorization.

![Graph](image)

Figure 5.5: FAR bound versus DR for State 2 of feature 4.

However, given a categorization, one can fine tune the threshold, by adjusting the bound on FAR. For example, if the bound for state 2 of feature 4 is changed to 20% (i.e., belief values between percentiles 10 and 90 are considered normal range), the DR can be improved. Figure 5.5 shows the effect of varying the bound on FAR on
the observed DR for state 2 of feature 4. Note that DR can be improved with good performance to 98%. This shows that BANBAD has the potential to detect anomaly even for a specific state of a feature.

From Figure 5.3 and 5.4, we observe that the performance becomes better when the dimension is increased, from D1 to D3, and from D4 to D6; more the data we can collect, better the performance. Obviously, this is as expected.

In summary, although we used normal distribution with specific $\mu$ and $\sigma$ to show the FAR and DR performance of BANBAD, similar performance can be expected for other values of $\mu$ and $\sigma$ or other probability distributions of raw data. The next section considers some of these scenarios. As mentioned above, using the percentile technique applied in a manner similar to the normal distributional features, we can always bound the FAR to a predefined percentage; and by fine tuning the percentile for some specific states if necessary, we can also achieve high DR for all situations.

5.1.2 Gamma distribution

After testing symmetry distribution, like normal distribution, we test a non-symmetric distribution, gamma distribution. Feature $f1$ of normal profile is generated by $\alpha$ and $\beta$ where $\alpha \times \beta = 30$ and $\alpha \times \beta^2 = \left(\frac{20}{3}\right)^2$. Similarly, feature $f1$ of anomaly is generated by $\alpha$ and $\beta$ where $\alpha \times \beta = 18$, $27$, $33$ or $42$ and $\alpha \times \beta^2 = \left(\frac{20}{3}\right)^2$. Other
features $f_2$ to $f_6$ are generated using scheme 1. Feature 4 is the target feature for anomaly detection. The FAR and DR are plotted in Figure 5.6 for all the features.

![Figure 5.6: False alarm rate and detection rate for all features (gamma distribution).](image)

We categorize feature 4 by setting 3 states. Uniformly distributed (ud) forms the same state interval, and skewed distributed (sd) forms different state interval. From Figure 5.6, we observe FAR of all features to be almost 0%, while the DR of all features to be almost 100%.

5.1.3 Non-linear relationship features

In this study, datasets with features having a non-linear relationship are tested as follows. We assume that the first feature, $f_1$ follows normal distribution with
mean, $\mu = 30$ and variance, $\sigma^2 = \left(\frac{20}{3}\right)^2$, other features $f_2$ to $f_6$ are generated using the following scheme, Scheme 2:

$$
\begin{align*}
  f_2 & : 1/2 \times f_1 + \delta_1 \\
  f_3 & : 2 \times f_2 + \delta_2 \\
  f_4 & : f_2 \times f_3 + \delta_3 \\
  f_5 & : 1/2 \times f_4 + \delta_4 \\
  f_6 & : 1/3 \times f_5 + \delta_5
\end{align*}
$$

where each $\delta_i$ is iid standard normal (mean 0 and variance 1).

Feature 4 is the target feature for anomaly detection. The FAR and DR are plotted in Figure 5.7 for all the features.

![Graph

Figure 5.7: False alarm rate and detection rate for all features having non-linear relationship.
We categorize feature 4 by setting 3 states. Uniformly distributed (ud) forms the same state interval, and skewed distributed (sd) forms different state interval. Form Figure 5.7, we observe the FAR of all features to be almost 0%, while the DR of all features to be almost 100%.

5.2 Effect of Categorization

From subsection 5.1, recall that FAR can be bounded by a predefined threshold; however, DR is not “very good” for detecting anomaly in individual states. Effect of categorization of feature 4 is plotted in Figure 5.8. We use scheme 1 here.
Figure 5.8: Effect of categorization of feature 4.

From Figure 5.8, we observe that the DR for discrete states is unstable for BANBAD. Figure 5.9 is plotted to further explain such phenomenon. Scheme 2 is used as an example for feature 4.
Based on the histogram plotted in Figure 5.9, we observe that there is an obvious probability distribution difference between the training and testing dataset for the value range from 0 to 1300. This indicates that we can detect anomaly with good performance for that state if it is set within this range. On the contrary, for the value range from 1300 to 1800, the training and testing dataset are almost overlapped which indicates that anomaly would be miss-detected (i.e., not detected) if the state is set within that range. In fact, one has little control over the categorization because probability distribution can be any value range in the real world, and different probability distributions of the testing dataset have different overlapped ranges with that of the training dataset. That's why BANBAD is not applicable for anomaly detection for any individual state, and is applicable to features as a whole.
CHAPTER VI

BANBAD PERFORMANCE ON REAL DATASETS

We tested BANBAD for two datasets collected from specific ad hoc networks and those are discussed in this chapter.

6.1 Wireless Sensor Network (WSN)

The dataset is taken from Intel Lab Data [21]. Data is collected by 54 sensors deployed in the Intel Berkeley Research Lab between February 28th and April 5th 2004. Four features are selected and sensors collect data for temperature, humidity, light, and voltage once every 31 seconds. From this raw dataset, we select a whole month (March) data for testing a sensor's behavior, e.g., sensor 2. Light is the target feature for anomaly detection. The data collected around midnight everyday are used for training to generate a normal profile and testing FAR, and the data collected after 8am everyday are used for testing DR.

After some raw data manipulation, we have the range \([\text{min}, \text{max}]\) of all 4 features of WSN as shown in Table 6.1.
Table 6.1: Range of 4 features of WSN

<table>
<thead>
<tr>
<th>Temperature</th>
<th>Humidity</th>
<th>Light</th>
<th>Voltage</th>
</tr>
</thead>
<tbody>
<tr>
<td>[18, 123)</td>
<td>[-4, 50)</td>
<td>[0, 626)</td>
<td>[2, 3)</td>
</tr>
</tbody>
</table>

By default, we use the following categorization scheme for 3 states:

Maximum value for state 1 is: \((\text{max} - \text{min}) \times 0.25 + \text{min}\)

Maximum value for state 2 is: \((\text{max} - \text{min}) \times 0.75 + \text{min}\),

Therefore, we have the categories of 4 features of WSN as shown in Table 6.2.

Table 6.2: Categories of 4 features of WSN

<table>
<thead>
<tr>
<th>Temperature</th>
<th>Humidity</th>
<th>Light</th>
<th>Voltage</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;51.25</td>
<td>&lt;9.5</td>
<td>&lt;156.25</td>
<td>&lt;2.25</td>
</tr>
<tr>
<td>[51.25, 96.75)</td>
<td>[9.5, 36.5)</td>
<td>[156.25, 469.5)</td>
<td>[2.25, 2.75)</td>
</tr>
<tr>
<td>&gt;=96.75</td>
<td>&gt;=36.5</td>
<td>&gt;=469.5</td>
<td>&gt;=2.75</td>
</tr>
</tbody>
</table>

By applying BANBAD, we obtain the following results as shown in Table 6.3.

Table 6.3: Anomaly sample # and performance of WSN

<table>
<thead>
<tr>
<th>Anomaly sample #</th>
<th>TWS</th>
<th>DR</th>
</tr>
</thead>
<tbody>
<tr>
<td>28</td>
<td>3.45%</td>
<td>96.55%</td>
</tr>
<tr>
<td>19</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

When we check anomaly sample #28, we find that some values of the light value are too high which indicates that there is an anomaly during the midnight; and when we check anomaly sample #19, we find that some values of the light value are too low which indicates that there is an anomaly during the morning. Here, real dataset does demonstrate normal behavior during certain time period; therefore, we intentionally select our target feature, light, between two different time periods,
around midnight and after 8am from a sensor, for anomaly detection. From Table 6.3, we observe that BANBAD exhibits excellent performance, in terms of both the false alarm rate and detection rate, for wireless sensor network.

### 6.2 Wireless Network (WN)

This dataset is taken from UMASS Trace Repository [73]. The data is collected for 60 laptop users to show their battery usage. From the raw dataset, six features are selected for our anomaly detection experiment. They are:

a) Battery capacity remaining (BCR)
b) Whether the machine was on AC or not (AC)
c) CPU utilization (CPU)
d) What was the disk space available (in MB) in the user account (DSA)
e) Whether the machine had Internet connectivity (INTERNET)
f) What was the time since there was a keyboard event (idle time) in milliseconds (IDLE),

BANBAD is selected to test a user's normal behavior, e.g., idle time for user #59. Our purpose is to demonstrate that a specific user has different idle time between different time periods. As such feature f) IDLE is our target feature for anomaly detection. The data collected around midnight from Sunday to Thursday are used for training to generate a normal profile and testing FAR, and the data collected after 2pm from Monday to Friday are used for testing DR.

After some raw data manipulation, we have the range \([\text{min}, \text{max})\) of all 6
features of WN as shown in Table 6.4.

Table 6.4: Range of 6 features of WN

<table>
<thead>
<tr>
<th></th>
<th>AC</th>
<th>CPU</th>
<th>DSA</th>
<th>INTERNET</th>
<th>IDLE</th>
</tr>
</thead>
<tbody>
<tr>
<td>BCR</td>
<td>[64, 100)</td>
<td>[0, 1]</td>
<td>[0, 95.31]</td>
<td>[55068, 69871]</td>
<td>[0, 1)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>[0, 438402657)</td>
</tr>
</tbody>
</table>

By using the default categorization scheme defined in WSN subsection for 3 states, we have the categories of 6 features of WN as shown in Table 6.5.

Table 6.5: Categories of 6 features of WN

<table>
<thead>
<tr>
<th></th>
<th>AC</th>
<th>CPU</th>
<th>DSA</th>
<th>INTERNET</th>
<th>IDLE</th>
</tr>
</thead>
<tbody>
<tr>
<td>BCR</td>
<td>&lt;73</td>
<td>&lt;23.83</td>
<td>&lt;58768.8</td>
<td>&lt;0.25</td>
<td>&lt;109600664.3</td>
</tr>
<tr>
<td></td>
<td>[73, 91)</td>
<td>[23.83, 71.48)</td>
<td>[58768.8, 66170.3)</td>
<td>[0.25, 0.75)</td>
<td>[109600664.3, 328801992.8)</td>
</tr>
<tr>
<td></td>
<td>&gt;=91</td>
<td>&gt;=71.48</td>
<td>&gt;=66170.3</td>
<td>&gt;=0.75</td>
<td>&gt;=328801992.8</td>
</tr>
</tbody>
</table>

By applying BANBAD, we obtain the following results as shown in Table 6.6.

Table 6.6: Anomaly sample # and performance of WN

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>DR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Anomaly sample #</td>
<td>N/A</td>
<td>1,2,4,6,7,9,10,11,12,13,16,19,20,21,22,24,25,26</td>
</tr>
<tr>
<td></td>
<td>0%</td>
<td>30.77%</td>
</tr>
</tbody>
</table>

From Table 6.6, we observe that DR is very low. By adjusting state 2 of feature f) IDLE to [1096006.6, 328801992.8), we obtain the updated results as shown in Table 6.7.

Table 6.7: Anomaly sample # and performance of WN

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>DR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Anomaly sample #</td>
<td>5</td>
<td>16</td>
</tr>
</tbody>
</table>
When we check anomaly sample #5, we find that some values of the idle time are too small which indicates that there is an anomaly during the midnight; and when we check anomaly sample #16, we find that some values of the idle time are too big which indicates that there is an anomaly during the workday (Monday to Friday, after 2pm). From Table 6.7, we observe that we achieve good performance in terms of false alarm rate and detection rate for the target feature, idle time, after adjusting categorization, for wireless network. Recall that one has little control over the categorization; therefore, it is not necessary to achieve good performance at once. From both Table 6.3 and 6.7, we demonstrate that BANBAD is widely applicable under different scenarios.

Known research results for IDS that use different types of systems and networks with different assumptions and different real datasets typically have FAR rates in the range [4%-6%] and DR rates in the range [90%-95%] [18-19]. Compared to those performances, our rates are slightly better.
CHAPTER VII

BANBAD TOOLKIT

7.1 Purpose

BANBAD toolkit is a console based software implementation of the BANBAD technique developed in Microsoft-Windows environment for anomaly detection. It is developed with few interactive steps to allow one to test any dataset based on beliefs.

7.2 Organization

The general structure of the BANBAD toolkit is depicted in Figure 7.1.

Figure 7.1: General structure of the BANBAD toolkit.
The general structure of BANBAD toolkit as shown in Figure 7.1 has two interactive steps, module categorization and module structure learning. Since one has little control over the categorization, therefore, interactive steps should be involved for categorization module during the training process. As to structure learning, it is out of the scope of this work, we simply use some existing, well-cited work for this module.

Therefore, by applying top-down design strategy, we have the following organization of BANBAD toolkit as shown in Figure 7.2.

![Diagram of BANBAD Toolkit]

Figure 7.2: Organization of the BANBAD toolkit.

The organization of BANBAD toolkit is depicted in Figure 7.2. Under "BANBAD Toolkit", the first level includes "Dataset", "exampleDataset", "BANBAD Software", "readme.txt", "user-manual.doc", and "BANBAD.BAT".
"exampleDataset" includes "DatasetInfo.txt", and raw dataset for training and testing processes, separately. "BANBAD Software" includes the modules of both the "Training process" and "Testing Process". "readme.txt" specifies the essential instructions for BANBAD toolkit to run, "user-manual.doc" gives the tutorial of how to use BANBAD toolkit, and "BANBAD.BAT" is simply for starting the BANBAD toolkit. Under "Training Process" module, we have three sub-modules for executing different tasks, "Categorization", "Sample Combiner", and "Belief Computation"; similarly, under "Testing Process" module, we have two sub-modules for executing different tasks, "Categorization" and "Belief Computation".

7.3 HW/SW Platforms

Currently, the BANBAD toolkit can be run on the MS-Windows platform and all executable files are either ".bat" or ".exe". All modules under training and testing processes are developed using visual C#, therefore we need MS Visual Studio 2008, or at least a C# IDE for editing and compiling the source code.

7.4 Installation

For instance, we install the BANBAD toolkit on the windows desktop (we assume "C" drive here, and the login name is "Administrator", e.g., "C:\Documents and Settings\Administrator\Desktop", we denote such desktop path as "PATH"). We generate a "BANBAD Toolkit" directory, and copy "Dataset", "exampleDataset", "BANBAD Software", "readme.txt", "user-manual.doc", and BANBAD.BAT into it.
Then we set the paths as follows:

In “BANBAD.bat”, set the path as:

```
DirPath="PATH\BANBAD Toolkit\"
```

In “BANBAD Training.bat”,

```
DirPath="PATH\BANBAD Toolkit\"
```

And In “BANBAD Testing.bat”,

```
Desktop="PATH"
```

```
DirPath="PATH\BANBAD Toolkit\"
```

### 7.5 Main Functions/Classes of BANBAD Software

In “Categorization” class, we have “categorization” function as shown in Figure 7.3.

- **Class:** Categorization
- **Function:** categorization
- **Parameters:** int NumberOfFeatures, double[,] Range, int[] NumberOfStates, double[,] States, double[] IntermediateState, int MaxNumberOfStates
- **Input:** number of features, range of each feature, maximum number of states
- **Output:** number of states/upper bound of state of each feature

Figure 7.3: Function of “categorization”.
In "BeliefComputation" class, we have "causalReasoningComputation", "diagnosticReasoningComputation", and "beliefComputation" functions as shown in Figure 7.4 to 7.6.

**Class:** BeliefComputation  
**Function:** causalReasoningComputation  
**Parameters:** 
- int NumberOfFeatures, int MaxNumberOfStates, int[] NumberOfRows, int[] NumberOfStates, int[,] CellValues, int Sample, double[,] Pi, StreamWriter PiRecord, StreamWriter EvaluatePiRecord, double[,] TempPi, double[,] Total, int targetFeature  
**Input:** number of features, maximum number of states, sample size, number of states of each feature, current sample  
**Output:** causal reasoning of the target feature

Figure 7.4: Function of "causalReasoningComputation".

**Class:** BeliefComputation  
**Function:** diagnosticReasoningComputation  
**Parameters:** 
- int NumberOfSamples, int NumberOfFeatures, int[] NumberOfStates, double[,] Mean, double[,] Pi, StreamWriter EvaluateLambdaRecord, double[,] Lambda, int targetFeature  
**Input:** number of features, maximum number of states, sample size, number of states of each feature, current sample  
**Output:** causal reasoning of the target feature

Figure 7.5: Function of "diagnosticReasoningComputation".
**Class:** BeliefComputation  
**Function:** beliefComputation  
**Parameters:** int NumberOfSamples, int NumberOfFeatures, int[] NumberOfStates, double[,] Pi, double[,] Lambda, double[,] Sigma, double[,] Belief, StreamWriter EvaluateBeliefRecord, int targetFeature  
**Input:** number of samples, number of features, number of states of each feature, pi message, lambda message, sigma message  
**Output:** belief of the target feature

Figure 7.6: Function of “beliefComputation”.

### 7.6 Tutorial

Real dataset of wireless network (WN) in subsection 6.2 is used for tutorial, feature 6 (IDLE time) is set to be the target feature for anomaly detection. We show various steps for an example run below. Executed from “BANBAD.BAT”, 
Step 1: Window popup as shown in Figure 7.7, waiting for loading raw training samples.

Figure 7.7: Waiting for loading raw dataset.
**Step 2:** After loading raw dataset, waiting for categorization as shown in Figure 7.8.

Figure 7.8: Waiting for categorization.
Step 3: We categorize all features as shown in Figure 7.9 for setting the target feature, states, and the threshold for FAR.

Summary of Dataset:
- Number of samples: 26
- Number of features: 6
- Maximum number of states: 18

Enter the target feature for anomaly detection (1-6): 6

Categorization of all features:

The range of feature 1 is: (64, 100)
Enter number of states of feature 1: 3
Upper bound for state 1 of feature 1: 73
Upper bound for state 2 of feature 1: 91

Feature 1 states: 
state1 [64, 73] state2 [73, 91] state3 [91, 100]

The range of feature 2 is: (0, 1)
Enter number of states of feature 2: 2
Upper bound for state 1 of feature 2: 0.4

Feature 2 states: 
state1 [0, 0.4] state2 [0.4, 1]

The range of feature 3 is: (0, 95.31)
Enter number of states of feature 3: 3
Upper bound for state 1 of feature 3: 23.83
Upper bound for state 2 of feature 3: 71.48

Feature 3 states: 
state1 [0, 23.83] state2 [23.83, 71.48] state3 [71.48, 95.31]

The range of feature 4 is: (554868, 69871)
Enter number of states of feature 4: 3
Upper bound for state 1 of feature 4: 58768
Upper bound for state 2 of feature 4: 66170

Feature 4 states: 
state1 [554868, 58768] state2 [58768, 66170] state3 [66170, 69871]

The range of feature 5 is: (0, 1)
Enter number of states of feature 5: 2
Upper bound for state 1 of feature 5: 0.6

Feature 5 states: 
state1 (0, 0.6] state2 [0.6, 1]

The range of feature 6 is: (0, 438402657)
Enter number of states of feature 6: 3
Upper bound for state 1 of feature 6: 1976006
Upper bound for state 2 of feature 6: 328881992

Feature 6 states: 
state1 (0, 1976006] state2 [1976006, 328881992] state3 [328881992, 438402657]

Enter the threshold for False Alarm Rate (0-100): 5

Figure 7.9: Categorization.
Step 4: Waiting for structure learning after categorization.

Figure 7.10: Waiting for structure learning.
**Step 5:** A well-cited structure learning toolkit, GeNe & SMILE [65] is applied for learning the DAG topology (and thus belief network). The structure learned is shown in Figure 7.11.

![Figure 7.11: Structure learned.](image-url)
Step 6: After structure learning, waiting for computing and displaying causal reasoning, diagnostic reasoning, and belief.

Figure 7.12: Waiting for computing and displaying causal reasoning.
Step 7: The causal reasoning is shown in Figure 7.13. Note that the sum of each sample is 1.0. Each number stands for the frequency of the state of that sample.

![Feature: 6]

<table>
<thead>
<tr>
<th>Sample</th>
<th>State 1</th>
<th>State 2</th>
<th>State 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sample1</td>
<td>0.0000</td>
<td>1.0000</td>
<td>0.0000</td>
</tr>
<tr>
<td>Sample2</td>
<td>0.0000</td>
<td>1.0000</td>
<td>0.0000</td>
</tr>
<tr>
<td>Sample3</td>
<td>0.0000</td>
<td>1.0000</td>
<td>0.0000</td>
</tr>
<tr>
<td>Sample4</td>
<td>0.0000</td>
<td>1.0000</td>
<td>0.0000</td>
</tr>
<tr>
<td>Sample5</td>
<td>0.9000</td>
<td>0.1000</td>
<td>0.0000</td>
</tr>
<tr>
<td>Sample6</td>
<td>0.0000</td>
<td>1.0000</td>
<td>0.0000</td>
</tr>
<tr>
<td>Sample7</td>
<td>0.0000</td>
<td>1.0000</td>
<td>0.0000</td>
</tr>
<tr>
<td>Sample8</td>
<td>0.0000</td>
<td>1.0000</td>
<td>0.0000</td>
</tr>
<tr>
<td>Sample9</td>
<td>0.1000</td>
<td>0.9000</td>
<td>0.0000</td>
</tr>
<tr>
<td>Sample10</td>
<td>0.0000</td>
<td>1.0000</td>
<td>0.0000</td>
</tr>
<tr>
<td>Sample11</td>
<td>0.0000</td>
<td>1.0000</td>
<td>0.0000</td>
</tr>
<tr>
<td>Sample12</td>
<td>0.0000</td>
<td>1.0000</td>
<td>0.0000</td>
</tr>
<tr>
<td>Sample13</td>
<td>0.0000</td>
<td>0.0000</td>
<td>1.0000</td>
</tr>
<tr>
<td>Sample14</td>
<td>0.0000</td>
<td>1.0000</td>
<td>0.0000</td>
</tr>
<tr>
<td>Sample15</td>
<td>0.0000</td>
<td>1.0000</td>
<td>0.0000</td>
</tr>
<tr>
<td>Sample16</td>
<td>0.0000</td>
<td>1.0000</td>
<td>0.0000</td>
</tr>
<tr>
<td>Sample17</td>
<td>0.0000</td>
<td>1.0000</td>
<td>0.0000</td>
</tr>
<tr>
<td>Sample18</td>
<td>0.0000</td>
<td>0.0000</td>
<td>1.0000</td>
</tr>
<tr>
<td>Sample19</td>
<td>0.0000</td>
<td>1.0000</td>
<td>0.0000</td>
</tr>
<tr>
<td>Sample20</td>
<td>0.0000</td>
<td>1.0000</td>
<td>0.0000</td>
</tr>
<tr>
<td>Sample21</td>
<td>0.0000</td>
<td>1.0000</td>
<td>0.0000</td>
</tr>
<tr>
<td>Sample22</td>
<td>0.0000</td>
<td>1.0000</td>
<td>0.0000</td>
</tr>
<tr>
<td>Sample23</td>
<td>0.0000</td>
<td>1.0000</td>
<td>0.0000</td>
</tr>
<tr>
<td>Sample24</td>
<td>0.0000</td>
<td>1.0000</td>
<td>0.0000</td>
</tr>
<tr>
<td>Sample25</td>
<td>0.0000</td>
<td>1.0000</td>
<td>0.0000</td>
</tr>
<tr>
<td>Sample26</td>
<td>0.0000</td>
<td>1.0000</td>
<td>0.0000</td>
</tr>
</tbody>
</table>

Figure 7.13: Display causal reasoning.
Step 8: The diagnostic reasoning is shown in Figure 7.14. Each number stands for the frequency of the state of that sample compared to the frequency of the state of all the training samples.

Figure 7.14: Display diagnostic reasoning.
**Step 9:** The belief is shown in Figure 7.15. Note that the sum of each sample is 1.0 after normalizing.

![Figure 7.15: Display belief.](image)

**Feature: 6**

<table>
<thead>
<tr>
<th>Sample</th>
<th>state1</th>
<th>state2</th>
<th>state3</th>
</tr>
</thead>
<tbody>
<tr>
<td>sample1</td>
<td>0.0000</td>
<td>1.0000</td>
<td>0.0000</td>
</tr>
<tr>
<td>sample2</td>
<td>0.0000</td>
<td>1.0000</td>
<td>0.0000</td>
</tr>
<tr>
<td>sample3</td>
<td>0.0000</td>
<td>1.0000</td>
<td>0.0000</td>
</tr>
<tr>
<td>sample4</td>
<td>0.0000</td>
<td>1.0000</td>
<td>0.0000</td>
</tr>
<tr>
<td>sample5</td>
<td>0.9995</td>
<td>0.0005</td>
<td>0.0000</td>
</tr>
<tr>
<td>sample6</td>
<td>0.0000</td>
<td>1.0000</td>
<td>0.0000</td>
</tr>
<tr>
<td>sample7</td>
<td>0.0000</td>
<td>1.0000</td>
<td>0.0000</td>
</tr>
<tr>
<td>sample8</td>
<td>0.0000</td>
<td>1.0000</td>
<td>0.0000</td>
</tr>
<tr>
<td>sample9</td>
<td>0.2210</td>
<td>0.7790</td>
<td>0.0000</td>
</tr>
<tr>
<td>sample10</td>
<td>0.0000</td>
<td>1.0000</td>
<td>0.0000</td>
</tr>
<tr>
<td>sample11</td>
<td>0.0000</td>
<td>1.0000</td>
<td>0.0000</td>
</tr>
<tr>
<td>sample12</td>
<td>0.0000</td>
<td>1.0000</td>
<td>0.0000</td>
</tr>
<tr>
<td>sample13</td>
<td>0.0000</td>
<td>0.0000</td>
<td>1.0000</td>
</tr>
<tr>
<td>sample14</td>
<td>0.0000</td>
<td>1.0000</td>
<td>0.0000</td>
</tr>
<tr>
<td>sample15</td>
<td>0.0000</td>
<td>1.0000</td>
<td>0.0000</td>
</tr>
<tr>
<td>sample16</td>
<td>0.0000</td>
<td>1.0000</td>
<td>0.0000</td>
</tr>
<tr>
<td>sample17</td>
<td>0.0000</td>
<td>1.0000</td>
<td>0.0000</td>
</tr>
<tr>
<td>sample18</td>
<td>0.0000</td>
<td>0.0000</td>
<td>1.0000</td>
</tr>
<tr>
<td>sample19</td>
<td>0.0000</td>
<td>1.0000</td>
<td>0.0000</td>
</tr>
<tr>
<td>sample20</td>
<td>0.0000</td>
<td>1.0000</td>
<td>0.0000</td>
</tr>
<tr>
<td>sample21</td>
<td>0.0000</td>
<td>1.0000</td>
<td>0.0000</td>
</tr>
<tr>
<td>sample22</td>
<td>0.0000</td>
<td>1.0000</td>
<td>0.0000</td>
</tr>
<tr>
<td>sample23</td>
<td>0.0000</td>
<td>1.0000</td>
<td>0.0000</td>
</tr>
<tr>
<td>sample24</td>
<td>0.0000</td>
<td>1.0000</td>
<td>0.0000</td>
</tr>
<tr>
<td>sample25</td>
<td>0.0000</td>
<td>1.0000</td>
<td>0.0000</td>
</tr>
<tr>
<td>sample26</td>
<td>0.0000</td>
<td>1.0000</td>
<td>0.0000</td>
</tr>
</tbody>
</table>
Step 10: The normal range of the target feature is shown in Figure 7.16.

![Training_range - Notepad](image)

<table>
<thead>
<tr>
<th>Feature: 6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Range of state 1: [0.0000, 0.2210]</td>
</tr>
<tr>
<td>Range of state 2: [0.0000, 1.0000]</td>
</tr>
<tr>
<td>Range of state 3: [0.0000, 1.0000]</td>
</tr>
</tbody>
</table>

Figure 7.16: Normal range of the target feature.

Step 11: after above 10 steps, the training process is done, the message is pop up as shown in Figure 7.17, waiting to start the testing process.

![C:\WINDOWS\system32\cmd.exe](image)

"Training completed! Ready to start testing"
Press any key to continue...

Figure 7.17: Waiting to start testing process.
Step 12: The belief values of the target feature for FAR is shown in Figure 7.18.

Figure 7.18: Compute and display belief values of the target feature for FAR.
Step 13: The belief values of the target feature for DR is shown in Figure 7.19.

Figure 7.19: Compute and display belief values of the target feature for DR.
Step 14: The statistical results of testing are shown in Figure 7.20 and 7.21, for FAR and DR, respectively. Based on the decision rule and the belief range shown in step 10, sample 5 is detected as an anomaly, it is false alarmed; sample 16 is miss-detected (i.e., sample 16 is an anomaly but not detected using BANBAD). From the last two lines in Figures 7.20 and 7.21, we observe that the false alarm rate and detection rate, for the target feature 6, are 3.846% and 96.154%, respectively.

```
Cumulative Results - Notepad
File Edit Format View Help
sample 1: normal
sample 2: normal
sample 3: normal
sample 4: normal
sample 5: feature 6 is an anomaly!!!
sample 6: normal
sample 7: normal
sample 8: normal
sample 9: normal
sample 10: normal
sample 11: normal
sample 12: normal
sample 13: normal
sample 14: normal
sample 15: normal
sample 16: normal
sample 17: normal
sample 18: normal
sample 19: normal
sample 20: normal
sample 21: normal
sample 22: normal
sample 23: normal
sample 24: normal
sample 25: normal
sample 26: normal

f1: 3 [ 73; 91; ]
f2: 2 [ 0.4; ]
f3: 3 [ 23.83; 71.48; ]
f4: 3 [ 58768; 66170; ]
f5: 2 [ 0.6; ]
f6: 3 [ 1096006; 328801992; ]

FAR: f6: 3.846%
```

Figure 7.20: False alarm rate of the running example.
<table>
<thead>
<tr>
<th>Sample</th>
<th>Feature 6</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Sample 1</td>
<td>feature 6 is normal</td>
<td></td>
</tr>
<tr>
<td>Sample 2</td>
<td>feature 6 is normal</td>
<td></td>
</tr>
<tr>
<td>Sample 3</td>
<td>feature 6 is normal</td>
<td></td>
</tr>
<tr>
<td>Sample 4</td>
<td>feature 6 is normal</td>
<td></td>
</tr>
<tr>
<td>Sample 5</td>
<td>feature 6 is normal</td>
<td></td>
</tr>
<tr>
<td>Sample 6</td>
<td>feature 6 is normal</td>
<td></td>
</tr>
<tr>
<td>Sample 7</td>
<td>feature 6 is normal</td>
<td></td>
</tr>
<tr>
<td>Sample 8</td>
<td>feature 6 is normal</td>
<td></td>
</tr>
<tr>
<td>Sample 9</td>
<td>feature 6 is normal</td>
<td></td>
</tr>
<tr>
<td>Sample 10</td>
<td>feature 6 is normal</td>
<td></td>
</tr>
<tr>
<td>Sample 11</td>
<td>feature 6 is normal</td>
<td></td>
</tr>
<tr>
<td>Sample 12</td>
<td>feature 6 is normal</td>
<td></td>
</tr>
<tr>
<td>Sample 13</td>
<td>feature 6 is normal</td>
<td></td>
</tr>
<tr>
<td>Sample 14</td>
<td>feature 6 is normal</td>
<td></td>
</tr>
<tr>
<td>Sample 15</td>
<td>feature 6 is normal</td>
<td></td>
</tr>
<tr>
<td>Sample 16</td>
<td>anomaly</td>
<td></td>
</tr>
<tr>
<td>Sample 17</td>
<td>feature 6 is normal</td>
<td></td>
</tr>
<tr>
<td>Sample 18</td>
<td>feature 6 is normal</td>
<td></td>
</tr>
<tr>
<td>Sample 19</td>
<td>feature 6 is normal</td>
<td></td>
</tr>
<tr>
<td>Sample 20</td>
<td>feature 6 is normal</td>
<td></td>
</tr>
<tr>
<td>Sample 21</td>
<td>feature 6 is normal</td>
<td></td>
</tr>
<tr>
<td>Sample 22</td>
<td>feature 6 is normal</td>
<td></td>
</tr>
<tr>
<td>Sample 23</td>
<td>feature 6 is normal</td>
<td></td>
</tr>
<tr>
<td>Sample 24</td>
<td>feature 6 is normal</td>
<td></td>
</tr>
<tr>
<td>Sample 25</td>
<td>feature 6 is normal</td>
<td></td>
</tr>
<tr>
<td>Sample 26</td>
<td>feature 6 is normal</td>
<td></td>
</tr>
</tbody>
</table>

f1: 3 [73; 91; ]
f2: 2 [0.4; ]
f3: 3 [23.83; 71.48; ]
f4: 3 [58768; 66170; ]
f5: 2 [0.6; ]
f6: 3 [1096006; 328801992; ]

DR: f6: 96.154%

Figure 7.21: Detection rate of the running example.

From Figure 7.20 and 7.21, the middle entries, such as f1: 3 [ 73; 91; ] displays the categorization of the first feature, which means the first feature has three
discrete states, the range of the first state is < 73, the range of the second state is [73, 91), and the range of the third state is ≥ 91.

**Step 15:** The last screenshot is shown in Figure 7.22 to indicate one run of the BANBAD toolkit is completed.

![Screenshot](image)

Figure 7.22: Final screen when the BANBAD toolkit completes one run.

Note that running BANBAD toolkit typically takes less than 1 minute for both training and testing processes to complete with a sample size of 1000 and 100 samples, tested on Intel(R) Core(TM)2 Duo CPU T7300 @ 2.00GHz, 2GB running Windows XP.
CHAPTER VIII

SUMMARY AND FUTURE WORK

We showed that MPB anomaly detection algorithm is effective for mobile networks. Each node's normal mobility profile is modeled as a Multi-Leaf tree structure. Clusters, generated through data mining, and the corresponding pattern strings, generated through fuzzy logic, are the two fundamental elements of the Multi-Leaf tree structure. Our proposed algorithm is then developed to detect potential internal attackers - masquerades in particular. Simulation results demonstrate that our proposed algorithm can achieve desirable performance in terms of both the false alarm rate and the detection rate for nodes with regular movement behaviors through a fine tuning the design parameter-threshold. Efficient fine tuning of this algorithm provides success and, subsequently, a basis for better anomaly detection in all mobile networks.

The novelty of MPB technique lies in easily generating and maintaining the normal profile; solving the more challenging problem of handling many close points and mobility patterns; achieving fairly good performance without strong assumptions and finally determining the design parameter, threshold, efficiently.

We next developed BANBAD and showed it to be an efficient anomaly detection technique based on belief networks that can handle incomplete samples and
multiple features. It has the potential to achieve high detection rates while reducing false alarm rates. Note that while our examples used energy consumption as the target feature for anomaly detection, our technique can be used to detect anomaly in any feature. It is not restricted to using energy consumption as the target feature for anomaly detection.

Our BANBAD technique is also based on statistical approach and significantly contributes to the field of anomaly detection in a few ways. First, it describes a method of easily generating and maintaining a profile; second, it achieves both high detection rate (≥95%) and low false alarm rate (≤5%) for the target feature, and false alarm rate can be bounded by certain predefined threshold; third, BANBAD is tested under many different statistical distributions, not limited to normal distributed dataset; fourth, it is widely applicable under different network scenarios, not limited to a specific network, e.g., ad hoc network, wireless sensor network, wireless network, etc.; fifth, BANBAD has the potential to function with an incomplete sample in the testing process, this function is useful in ad hoc networks, because its dynamically changing topology can result in the incomplete observations for the selected features; sixth BANBAD exhibits excellent performance on the two real datasets we used, in addition to achieving good performance, it detect anomaly efficiently and accurately; seventh, the BANBAD toolkit developed allows one to use it for anomaly detection easily and effectively.

There is, obviously, potential for future work. Recall that the centralized BANBAD addressed in chapter IV, we have a one time training process to generate a
normal profile; however, to keep the normal profile updated and use it for further testing, it's good to design an adaptive learning technique for BANBAD. Intuitively, the more the information gain, the better the performance. The challenge lies in: how to take the error accumulation into account. Here, error could be defined as the summation of the false alarm rate (FAR) and miss-detection rate (1-DR), and the goal is to bound the error while involving more data for training. Based on the sequential hypothesis testing [74], such error should be theoretically bounded; otherwise, we reach the stopping point, and need to check the machine and/or data collection errors.

Furthermore, distributed BANBAD could be developed for both the BANBAD processes (training and testing), data and features collection, etc.; seasonal effect, time series technique could be involved for users to select different time periods for anomaly detection, not just from human intuition and/or personal experience; feature selection technique could be explored under different network scenarios for anomaly detection, intuitively, the more the relationship among the features, the better the performance of BANBAD, especially for testing incomplete samples; and structure learning technique could be developed under the following minimum requirements: it integrates many well cited structure learning algorithms; it can test both discrete and continuous datasets; and it has the functionality of console debugging. Therefore, BANBAD toolkit can be upgraded to fully automated version by smoothly incorporating such structure learning techniques, and effectively test any dataset.
Appendix A

BANBAD Toolkit Source Code

C# source codes of various BANBAD Toolkit modules are listed here.

/* Categorization_training: Categorize discrete states of all features
 * Author: Chaoli Cai
 * Input: DatasetInfo.txt, n samples (raw data)
 * Output: record_from_discretized.txt, statesFromTraining.txt,
 * subfolder (discrete_sample0_serial,
 * discrete_sample1_serial, ...)
 *
 * Modified: 11/20/09
 * */

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Text.RegularExpressions;
using System;

namespace Categorization
{
    class Categorization
    {
        static void Main(string[] args)
        {
            // read info. from DatasetInfo.txt
            StreamReader sr = new StreamReader("DatasetInfo.txt");
            int NumberOfSamples = Convert.ToInt32(sr.ReadLine());
            int NumberOfFeatures = Convert.ToInt32(sr.ReadLine());
            int MaxNumberOfStates = Convert.ToInt32(sr.ReadLine());

            // output summary to console

            // ...
Console.WriteLine("Summary of Dataset: ");
Console.WriteLine("# of samples: {0}", NumberOfSamples);
Console.WriteLine("# of features: {0}", NumberOfFeatures);
Console.WriteLine("Max # of states: {0}", MaxNumberOfStates);

// enter the target feature for anomaly detection
StreamWriter tarFeature = new StreamWriter("targetFeature.txt");
string readTargetFeature;
targetFeature;
while (true)
{
    Console.Write("\nEnter the target feature for anomaly detection
" + (0) + ", (1), (NumberofFeatures);"
    readTargetFeature = Console.ReadLine();
    if (IsUint(readTargetFeature) &&
            (Convert.ToInt32(readTargetFeature) <= NumberOfFeatures
                && Convert.ToInt32(readTargetFeature) >= (1))
    {
        targetFeature = Convert.ToInt32(readTargetFeature);
        break;
    }
}
tarFeature.WriteLine(targetFeature);
tarFeature.Close();

Console.WriteLine("\nCategorization of all features ");

// get the range for each feature
double[,] Range = new double[NumberOfFeatures, 2];
for (int i = 0; i < NumberOfFeatures; i++)
{
    Range[i, 0] = Convert.ToDouble(sr.ReadLine());
    Range[i, 1] = Convert.ToDouble(sr.ReadLine());
}

int[,] NumberOfStates = new int[NumberOfFeatures];
double[,] , , States = new double[NumberOfFeatures, MaxNumberOfStates, 2];

// initialize state j of feature i
for (int i = 0; i < NumberOfFeatures; i++)
{
for (int j = 0; j < MaxNumberOfStates; j++)
{
    States[i, j, 0] = 0;
    States[i, j, 1] = 0;
}
}
Console.WriteLine();

double[] IntermediateState = new double[NumberOfFeatures];
for (int i = 0; i < NumberOfFeatures; i++)
    IntermediateState[i] = 0;

// setup states for each feature
categorization(NumberOfFeatures, Range, NumberOfStates, States,
IntermediateState, MaxNumberOfStates);

// setup threshold
double threshold;
while (true)
{
    Console.Write("Enter the threshold for False Alarm Rate (0-100): ");
    string thresh = Console.ReadLine();
    if (IsNumber(thresh))
    {
        threshold = Convert.ToDouble(thresh);
        break;
    }
}

// files operation
string currentDir = Directory.GetCurrentDirectory();
string NumOfStates_folder = "subfolder";
Directory.CreateDirectory(NumOfStates_folder);
StreamWriter rec = new StreamWriter("record_from_discretized.txt");
rec.WriteLine(NumberOfSamples);
rec.WriteLine(NumberOfFeatures);
rec.WriteLine(threshold);
for (int f = 0; f < NumberOfFeatures; f++)
{

rec.WriteLine(NumberOfStates[f]);
}
rec.WriteLine(MaxNumberOfStates);
rec.WriteLine(targetFeature);
rec.Close();

for (int d = 0; d < NumberOfSamples; d++)
{
    StreamReader sr_serial = new StreamReader("sample" +
        d.ToString() + "_serial.txt");
    int NumberOfRows = Convert.ToInt32(sr_serial.ReadLine());

    double[] value = new double[NumberOfRows *
        NumberOfFeatures];
    for (int i = 0; i < NumberOfRows; i++)
    {
        for (int j = 0; j < NumberOfFeatures; j++)
            value[i * NumberOfFeatures + j] =
                Convert.ToDouble(sr_serial.ReadLine());
    }

    Directory.SetCurrentDirectory(NumOfStates_folder);

    StreamWriter sw_serial = new StreamWriter("discrete_sample"
        + d.ToString() + "_serial.txt");
    StreamWriter sw = new StreamWriter("discrete_sample" +
        d.ToString() + ".txt");

    sw_serial.WriteLine("{0}", NumberOfRows);
    for (int i = 0; i < NumberOfFeatures; i++)
    {
        sw.Write("f" + (i + 1).ToString() + "\t");
    }

    for (int i = 0; i < NumberOfRows; i++)
    {
        for (int j = 0; j < NumberOfFeatures; j++)
        {
            bool founded = false;
            for (int k = 0; k < NumberOfStates[j] - 1; k++)
            {
                if ((value[i * NumberOfFeatures + j] >=
                    States[j, k, 0]) &
                    (value[i *
                        NumberOfFeatures + j] < States[j, k, 1]))
{ 
    sw_serial.WriteLine("{0} ", k + 1);
    if (j == NumberOfFeatures - 1)
        sw.Write("{0,-6}" , k + 1);
    else
        sw.Write("{0,-6}\t" , k + 1);
    founded = true;
    break;
}
if (founded == false)
{
    if ((valuep * NumberOfFeatures + j] <
        States[j, 0, 0])
    {
        sw_serial.WriteLine("{0} ", 1);
        if (j == NumberOfFeatures - 1)
            sw.Write("{0,-6}" , 1);
        else
            sw.Write("{0,-6}\t" , 1);
    }
    else
    {
        sw_serial.WriteLine("{0} ",
            NumberOfStates[j]);
        if (j == NumberOfFeatures - 1)
            sw.Write("{0,-6}" ,
                NumberOfStates[j]);
        else
            sw.Write("{0,-6}\t" ,
                NumberOfStates[j]);
    }
}
sw.WriteLine();
}
sw_serial.Close();
sw.Close();
Directory.SetCurrentDirectory(currentDir);
// type checking for numbers
public static bool IsNumber(string input)
{
    string pattern = "^-?\d+\d\d+\d\d+$|\d\d+\d\d\d\d+$";
    Regex regex = new Regex(pattern);
    return regex.IsMatch(input);
}

// type checking for positive integer numbers
public static bool IsUint(string input)
{
    Regex regex = new Regex("^[0-9]*[1-9][0-9]*$");
    return regex.IsMatch(input);
}

// categorization for each feature
public static void categorization(int NumberOfFeatures, double[,] Range, int[] NumberOfStates, double[,] States, double[] IntermediateState, int MaxNumberOfStates)
{
    StreamWriter statesFromTraining = new StreamWriter("statesFromTraining.txt");
    for (int i = 0; i < NumberOfFeatures; i++)
    {
        IntermediateState[i] = 0;
        if (i == 0)
            Console.WriteLine("The range of feature {0} is: [{1}, {2}]", i + 1, Range[i, 0], Range[i, 1]);
        else
            Console.WriteLine("The range of feature {0} is: [{1}, {2}]", i + 1, Range[i, 0], Range[i, 1]);
        // type checking for # of states of features (Integer type)
        while (true)
        {
            Console.Write("Enter # of states of feature "+(i + 1).ToString() + ": ");
            string inputOfStates = Console.ReadLine();
            if (IsUint(inputOfStates) && Convert.ToInt32(inputOfStates) <= MaxNumberOfStates))
NumberOfStates[i] =
Convert.ToInt32(inputOfStates);
break;
}

statesFromTraining.WriteLine(NumberOfStates[i]);
IntermediateState[i] = Range[i, 0];
for (int j = 0; j < NumberOfStates[i] - 1; j++)
{
    States[i, j, 0] = IntermediateState[i];
    // type checking for states (Double type)
    while (true)
    {
        Console.Write("Upper bound for state " + (j + 1).ToString() + " of feature " + (i + 1).ToString() + ": ");
        string state = Console.ReadLine();
        if (IsNumber(state))
        {
            States[i, j, 1] =
Convert.ToDouble(state);
            statesFromTraining.WriteLine(States[i, j, 1]);
            break;
        }
    }
}

while (true)
{
    if (((j != 0) && (States[i, j, 1] > States[i, j, 0]) && (States[i, j, 1] < Range[i, 1])) || ((j == 0) && (States[i, j, 1] >= States[i, j, 0]) && (States[i, j, 1] < Range[i, 1])))
    break;
else
{
    // type checking for states (Double type)
    while (true)
    {
Console.WriteLine("Upper bound for state " + (j + 1).ToString() + " of feature " + (i + 1).ToString() + ": ");
string temp = Console.ReadLine();
if (IsNumber(temp))
{
    States[i, j, 1] = Convert.ToDouble(temp);
    statesFromTraining.WriteLine(States[i, j, 1]);
    break;
}
}
IntermediateState[i] = States[i, j, 1];

// output discrete states settings to console
Console.WriteLine();
Console.WriteLine("feature {0} states: \t", (i + 1).ToString());
for (int j = 0; j <= NumberOfStates[i] - 1; j++)
{
    if (NumberOfStates[i] == 1)
    {
        Console.WriteLine("state{0} [{1}, {2}) ", (j + 1).ToString(), Range[i, 0], Range[i, 1]);
    }
    else if (j == NumberOfStates[i] - 1)
    {
        Console.WriteLine("state{0} [{1}, {2}) ", (j + 1).ToString(), IntermediateState[i], Range[i, 1]);
    }
    else
    {
        Console.WriteLine("state{0} [{1}, {2}) ", (j + 1).ToString(), States[i, j, 0], States[i, j, 1]);
    }
}
Console.WriteLine();
statesFromTraining.Close();}}
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.IO;

namespace SampleCombiner
{
    class SampleCombiner
    {
        static void Main(string[] args)
        {
            StreamReader sr = new StreamReader("record_from_discretized.txt");
            int NumberOfSamples = Convert.ToInt32(sr.ReadLine());
            int NumberOfFeatures = Convert.ToInt32(sr.ReadLine());

            // output table header
            StreamWriter sw = new StreamWriter("combined.txt");
            for (int i = 0; i < NumberOfFeatures - 1; i++)
            {
                sw.Write("f" + (i + 1).ToString() + "\t");
            }
            sw.Write("f" + (NumberOfFeatures).ToString() + "\n");
            sw.WriteLine();

            // combine samples after categorization
            for (int i = 0; i < NumberOfSamples; i++)
            {
            }
StreamReader sr_discrete_sample = new StreamReader("discrete_sample" + i.ToString() + "_serial.txt");
int cases = Convert.ToInt32(sr_discrete_sample.ReadLine());

for (int j = 0; j < cases; j++)
{
    for (int k = 0; k < NumberOfFeatures; k++)
    {
        if (k != NumberOfFeatures - 1)
            sw.Write("{0,-6}\t", Convert.ToInt32(sr_discrete_sample.ReadLine()));
        else
            sw.Write("{0,-6}", Convert.ToInt32(sr_discrete_sample.ReadLine()));
    }
    sw.WriteLine();
}
sw.Close();
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.IO;

namespace BeliefComputation
{
    class BeliefComputation
    {
        static void Main(string[] args)
        {
            // read info. (NumberOfSamples, NumberOfFeatures, threshold)
            // from record_from_discretized.txt
            StreamReader sr_record = new StreamReader("record_from_discretized.txt");
            int NumberOfSamples = Convert.ToInt32(sr_record.ReadLine());
            int NumberOfFeatures = Convert.ToInt32(sr_record.ReadLine());
            double threshold = Convert.ToDouble(sr_record.ReadLine());

            // read info. from targetFeature.txt
            StreamReader tf = new StreamReader("targetFeature.txt");
            int targetFeature = Convert.ToInt32(tf.ReadLine());

            StreamWriter PiRecord = new StreamWriter("pi_record.txt");
            StreamWriter EvaluatePiRecord = new StreamWriter("EvaluatePi.txt");
        }
    }
}

/* BeliefComputation_training: Compute belief ranges for all features
during the training process
*
* Author: Chaoli Cai
* Input: record_from_discretized.txt, subfolder
discrete_sample0_serial, discrete_sample1_serial, ...
* Output: belief ranges for all features
* Modified: 11/20/09
* */
EvaluatePiRecord.WriteLine("Feature: " +
targetFeature.ToString());
StreamWriter EvaluateLambdaRecord = new
StreamWriter("EvaluateLambda.txt");
EvaluateLambdaRecord.WriteLine("Feature: " +
targetFeature.ToString());
StreamWriter EvaluateSigmaRecord = new
StreamWriter("EvaluateSigma.txt");
EvaluateSigmaRecord.WriteLine("Feature: " +
targetFeature.ToString());
StreamWriter EvaluateBeliefRecord = new
StreamWriter("EvaluateBelief.txt");
EvaluateBeliefRecord.WriteLine("Feature: " +
targetFeature.ToString());

StreamWriter Training_range = new
StreamWriter("Training_range.txt");
StreamWriter Training_range_serial = new
StreamWriter("Training_range_serial.txt");

int[] NumberOfStates = new int[NumberOfFeatures];
for (int i = 0; i < NumberOfFeatures; i++)
{
    NumberOfStates[i] =
Convert.ToInt32(sr_record.ReadLine());
}

int MaxNumberOfStates =
Convert.ToInt32(sr_record.ReadLine());

int NumberOfDatasets = 0;
StreamReader[] sr_file = new
StreamReader[NumberOfSamples];
int[] NumberOfRows = new int[NumberOfSamples];
int TotalCases = 0;

double[,] TempPi = new double[NumberOfFeatures,
MaxNumberOfStates];
double[,] Total = new double[NumberOfFeatures,
MaxNumberOfStates];
double[,] Mean = new double[NumberOfFeatures,
MaxNumberOfStates];
double[,] Lambda = new double[NumberOfSamples, NumberOfFeatures, MaxNumberOfStates];
double[,] Belief = new double[NumberOfSamples, NumberOfFeatures, MaxNumberOfStates];
double[,] Pi = new double[NumberOfSamples, NumberOfFeatures, MaxNumberOfStates];
double[,] Sigma = new double[NumberOfSamples, NumberOfFeatures];
double[] SortBelief = new double[NumberOfSamples];

for (int r = 0; r < NumberOfFeatures; r++)
{
    for (int c = 0; c < MaxNumberOfStates; c++)
    {
        TempPi[r, c] = 0;
        Total[r, c] = 0;
        Mean[r, c] = 0;
    }
}

for (int d = 0; d < NumberOfSamples; d++)
{
    for (int r = 0; r < NumberOfFeatures; r++)
    {
        Sigma[d, r] = 0;
        for (int c = 0; c < MaxNumberOfStates; c++)
        {
            Pi[d, r, c] = 0;
            Lambda[d, r, c] = 0;
            Belief[d, r, c] = 0;
        }
    }
}

for (int i = 0; i < NumberOfFeatures; i++)
{
    if ((i + 1) == targetFeature)
    {
        for (int c = 0; c < NumberOfStates[i]; c++)
        {
            if (c == 0)
            {

EvaluatePiRecord.Write("\t\tstate{0} \\
, (c + 1).ToString());
EvaluateLambdaRecord.Write("\t\tstate{0}", (c + 1).ToString());
EvaluateBeliefRecord.Write("\t\tstate{0} \\
, (c + 1).ToString());
}
else
{
    EvaluatePiRecord.Write("\t\tstate{0} \\
, (c + 1).ToString());
    EvaluateLambdaRecord.Write("\tstate{0} \\
, (c + 1).ToString());
    EvaluateBeliefRecord.Write("\tstate{0} \\
, (c + 1).ToString());
}
}
EvaluatePiRecord.WriteLine();
EvaluateLambdaRecord.WriteLine();
EvaluateBeliefRecord.WriteLine();

for (int Sample = 0; Sample < NumberOfSamples; Sample++)
{
    if (File.Exists("discrete_sample + Sample.ToString() + \\
".serial.txt"))
    {
        NumberOfDatasets++;
        sr_file[Sample] = new StreamReader("discrete_sample + Sample.ToString() + \\
".serial.txt");
        NumberOfRows[Sample] = Convert.ToInt32(sr_file[Sample].ReadLine());
        TotalCases = TotalCases + \\
NumberOfRows[Sample];
        int[,] CellValues = new int[NumberOfRows[Sample], NumberOfFeatures];
        for (int row = 0; row < NumberOfRows[Sample]; row++)
        {
            }
for (int col = 0; col < NumberOfFeatures; col++)
{
    CellValues[row, col] = Convert.ToInt32(sr_file[Sample].ReadLine());
}

// compute causal reasoning
causalReasoningComputation(NumberOfFeatures,
MaxNumberOfStates, NumberOfRows, NumberOfStates,
CellValues, Sample, Pi, PiRecord, EvaluatePiRecord, TempPi,
Total, targetFeature);
}
else
    continue;
}
PiRecord.Close();

string answer = "training";
if (answer == "training")
{
    StreamWriter mean_Pi_record = new StreamWriter("mean_Pi_record.txt");
    for (int r = 0; r < NumberOfFeatures; r++)
    {
        if ((r + 1) == targetFeature)
        {
            for (int c = 0; c < NumberOfStates[r]; c++)
            {
                Mean[r, c] = Total[r, c] / NumberOfSamples;
                mean_Pi_record.WriteLine("{0:f4}\t", Mean[r, c]);
            }
        }
    }
    mean_Pi_record.Close();
}
EvaluatePiRecord.Close();
// get mean of causalReasoning
StreamReader sr = new StreamReader("mean_PLrecord.txt");
for (int r = 0; r < NumberOfFeatures; r++)
{
    if ((r + 1) == targetFeature)
    {
        for (int c = 0; c < NumberOfStates[r]; c++)
            Mean[r, c] = Convert.ToDouble(sr.ReadLine());
    }
}

// compute diagnostic reasoning
diagnosticReasoningComputation(NumberOfSamples,
NumberOfFeatures, NumberOfStates, Mean, Pi,
EvaluateLambdaRecord, Lambda, targetFeature);
EvaluateLambdaRecord.Close();

// compute sigma
sigmaComputation(NumberOfSamples, NumberOfFeatures,
NumberOfStates, Pi, Lambda, EvaluateSigmaRecord, Sigma,
targetFeature);
EvaluateSigmaRecord.Close();

// compute belief
beliefComputation(NumberOfSamples, NumberOfFeatures,
NumberOfStates, Pi, Lambda, Sigma, Belief,
EvaluateBeliefRecord, targetFeature);
EvaluateBeliefRecord.Close();

// get belief ranges for all features
for (int r = 0; r < NumberOfFeatures; r++)
{
    if ((r + 1) == targetFeature)
    {
        Training_range.WriteLine("Feature: "+
targetFeature.ToString());
        Training_range.WriteLine();
        for (int c = 0; c < NumberOfStates[r]; c++)
        {
            Training_range.Write("Range of state {0}: ", (c +
1).ToString());
        }
    }
for (int d = 0; d < NumberOfSamples; d++)
{
    SortBelief[d] = Belief[d, r, c];
}
SortandGetRanges(SortBelief);
Training_range.Write("[{0:f4}, {1:f4}]",
    SortBelief[(int)((double)(threshold) /
    (double)NumberOfStates[r])],
    SortBelief[(int)(NumberOfSamples -
    (double)(threshold) /
    (double)NumberOfStates[r])]);
Training_range.WriteLine();
Training_range_serial.WriteLine("{0:f4}",
    SortBelief[(int)((double)(threshold) /
    (double)NumberOfStates[r])]);
Training_range_serial.WriteLine("{0:f4}",
    SortBelief[(int)(NumberOfSamples -
    (double)(threshold) /
    (double)NumberOfStates[r])]);
}
Training_range.WriteLine();
}
Training_range.Close();
Training_range_serial.Close();

// sort belief values
static void SortandGetRanges(double[] SortBelief)
{
    for (int i = 0; i < SortBelief.Length; i++)
    {
        int s = i;
        for (int j = i + 1; j < SortBelief.Length; j++)
        {
            if (SortBelief[j] < SortBelief[s])
            {
                s = j;
            }
        }
    double t = SortBelief[i];
    SortBelief[i] = SortBelief[s];
SortBelief[s] = t;
}

// compute causal reasoning (pi)
static void causalReasoningComputation(int NumberOfFeatures, int MaxNumberOfStates, int[,] NumberOfRows, int[,] NumberOfStates, int[,] CellValues, int Sample, double[,] Pi, StreamWriter PiRecord, StreamWriter EvaluatePiRecord, double[,] TempPi, double[,] Total, int targetFeature)
{
    Console.WriteLine("\nsample " + Sample.ToString());
    int[,] freq = new int[NumberOfFeatures, MaxNumberOfStates];
    Console.WriteLine();
    for (int f = 0; f < NumberOfFeatures; f++)
    {
        if ((f + 1) == targetFeature)
        {
            if (Sample < 9)
                EvaluatePiRecord.Write("\nsample " + (Sample + 1).ToString() + "\t\t")
            else
                EvaluatePiRecord.Write("\nsample " + (Sample + 1).ToString() + "\t")
            int k = 0;
            for (int j = 0; j < NumberOfRows[Sample]; j++)
            {
                for (k = 0; k < NumberOfStates[f]; k++)
                {
                    if (CellValues[j, f] == k + 1)
                    {
                        freq[f, k]++;
                        break;
                    }
                }
            }
            Console.WriteLine("\nfeature " + (f + 1).ToString());
            for (k = 0; k < NumberOfStates[f]; k++)
            {
                TempPi[f, k] = (double)freq[f, k] / NumberOfRows[Sample];
            }
        }
    }
}
Total[f, k] = Total[f, k] + TempPi[f, k];
Console.WriteLine("\t\tstate " + (k + 1).ToString() + " pi: {0:f4} ", TempPi[f, k]);
PiRecord.WriteLine("{0:f4}" , TempPi[f, k]);
EvaluatePiRecord.Write("{0:f4}\t", TempPi[f, k]);
Pi[Sample, f, k] = TempPi[f, k];
}
EvaluatePiRecord.WriteLine();

// compute diagnostic reasoning (lambda)
static void diagnosticReasoningComputation(int NumberOfSamples, int NumberOfFeatures, int[] NumberOfStates, double[,] Mean, double[,] Pi, StreamWriter EvaluateLambdaRecord, double[,] Lambda, int targetFeature)
{
    for (int d = 0; d < NumberOfSamples; d++)
    {
        for (int r = 0; r < NumberOfFeatures; r++)
        {
            if ((r + 1) == targetFeature)
            {
                if (d < 9)
                    EvaluateLambdaRecord.Write("\n\t\tsample" + (d + 1).ToString() + "\t\t");
                else
                    EvaluateLambdaRecord.Write("\n\t\tsample" + (d + 1).ToString() + "\t");
            for (int c = 0; c < NumberOfStates[r]; c++)
            {
                if (Mean[r, c] == 0)
                {
                    Lambda[d, r, c] = 1;
                    EvaluateLambdaRecord.Write("{0:f4}\t", Lambda[d, r, c]);
                }
                else
                    
            }
        }
    }
}
\begin{verbatim}
Lambda[d, r, c] = Pi[d, r, c] / Mean[r, c];
EvaluateLambdaRecord.WriteLine("{0:f4}\t", Lambda[d, r, c]);
}
}
EvaluateLambdaRecord.WriteLine();

// compute sigma
static void sigmaComputation(int NumberOfSamples, int NumberOfFeatures, int[] NumberOfStates, double[,] Pi, double[,] Lambda, StreamWriter EvaluateSigmaRecord, double[,] Sigma, int targetFeature)
{
    for (int d = 0; d < NumberOfSamples; d++)
    {
        for (int r = 0; r < NumberOfFeatures; r++)
        {
            if ((r + 1) == targetFeature)
            {
                for (int c = 0; c < NumberOfStates[r]; c++)
                {
                    Sigma[d, r] = Sigma[d, r] + Pi[d, r, c] * Lambda[d, r, c];
                }
                EvaluateSigmaRecord.WriteLine("{0:f4}\t", Sigma[d, r]);
            }
        }
    }
}

// compute belief (by applying the Bayes's rule)
static void beliefComputation(int NumberOfSamples, int NumberOfFeatures, int[] NumberOfStates, double[,] Pi, double[,] Lambda, double[,] Sigma, double[,] Belief, StreamWriter EvaluateBeliefRecord, int targetFeature)
{

\end{verbatim}
for (int d = 0; d < NumberOfSamples; d++)
{
    for (int r = 0; r < NumberOfFeatures; r++)
    {
        if ((r + 1) == targetFeature)
        {
            if (d < 9)
                EvaluateBeliefRecord.Write("sample" + (d + 1).ToString() + "\t\t");
            else
                EvaluateBeliefRecord.Write("sample" + (d + 1).ToString() + "\t");
        }
    }
}

for (int c = 0; c < NumberOfStates[r]; c++)
{
    if (Sigma[d, r] == 0)
        Belief[d, r, c] = 1;
    else
        Belief[d, r, c] = Pi[d, r, c] * Lambda[d, r, c] / Sigma[d, r];
    EvaluateBeliefRecord.Write("{0:f4}\t", Belief[d, r, c]);
}

EvaluateBeliefRecord.WriteLine();
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System;

namespace Categorization
{
    class Categorization
    {
        static void Main(string[] args)
        {
            // read info. from DatasetInfo.txt
            StreamReader sr = new StreamReader("DatasetInfo.txt");
            int NumberOfSamples = Convert.ToInt32(sr.ReadLine());
            int NumberOfFeatures = Convert.ToInt32(sr.ReadLine());
            int MaxNumberOfStates = Convert.ToInt32(sr.ReadLine());

            // get the range for each feature
            double[,] Range = new double[NumberOfFeatures, 2];
            for (int i = 0; i < NumberOfFeatures; i++)
            {
                Range[i, 0] = Convert.ToDouble(sr.ReadLine());
                Range[i, 1] = Convert.ToDouble(sr.ReadLine());
            }

            int[] NumberOfStates = new int[NumberOfFeatures];
            double[, ,] States = new double[NumberOfFeatures, MaxNumberOfStates, 2];

            // initialize state j of feature i
        }
    }
}
for (int i = 0; i < NumberOfFeatures; i++)
{
    for (int j = 0; j < MaxNumberOfStates; j++)
    {
        States[i, j, 0] = 0;
        States[i, j, 1] = 0;
    }
}
Console.WriteLine();

double[] IntermediateState = new double[NumberOfFeatures];
for (int i = 0; i < NumberOfFeatures; i++)
IntermediateState[i] = 0;

// setup states for each feature
categorization(NumberOfFeatures, Range, NumberOfStates,
States, IntermediateState);

// files operation
string currentDir = Directory.GetCurrentDirectory();
string NumOfStates_folder = "subfolder";
Directory.CreateDirectory(NumOfStates_folder);
StreamWriter rec = new
StreamWriter("record_from_discretized.txt");
rec.WriteLine(NumberOfSamples);
rec.WriteLine(NumberOfFeatures);
for (int f = 0; f < NumberOfFeatures; f++)
{
    rec.WriteLine(NumberOfStates[f]);
}
rec.WriteLine(MaxNumberOfStates);
rec.Close();

for (int d = 0; d < NumberOfSamples; d++)
{
    StreamReader sr_serial = new StreamReader("sample" +
d.ToString() + ".serial.txt");
    int NumberOfRows =
Convert.ToInt32(sr_serial.ReadLine());
double[] value = new double[NumberOfRows * NumberOfFeatures];
for (int i = 0; i < NumberOfRows; i++)
    for (int j = 0; j < NumberOfFeatures; j++)
        value[i * NumberOfFeatures + j] = Convert.ToDouble(sr_serial.ReadLine());

Directory.SetCurrentDirectory(NumOfStates_folder);
StreamWriter sw_serial = new StreamWriter("discrete_sample" + d.ToString() + ".serial.txt");
StreamWriter sw = new StreamWriter("discrete_sample" + d.ToString() + ".txt");
sw_serial.WriteLine("0", NumberOfRows);
for (int i = 0; i < NumberOfFeatures - 1; i++)
{
    sw.Write("f" + (i + 1).ToString() + "	");
}
sw.Write("f" + (NumberOfFeatures).ToString());
sw.WriteLine();
for (int i = 0; i < NumberOfRows; i++)
{
    for (int j = 0; j < NumberOfFeatures; j++)
    {
        bool founded = false;
        for (int k = 0; k < NumberOStates[j] - 1; k++)
        {
            if ((value[i * NumberOfFeatures + j] >= States[j, k, 0]) && (value[i * NumberOfFeatures + j] < States[j, k, 1]))
            {
                sw_serial.WriteLine("{0} ", k + 1);
                if (j == NumberOfFeatures - 1)
                    sw.Write("{0,- 6}"", k + 1);
                else
sw.Write("{0,-6}\t", k + 1);
founded = true;
break;
}
}
if (founded == false)
{
    if ((value[i * NumberOfFeatures + j] < States[j, 0, 0]))
    {
        sw_serial.WriteLine("{0} ",
        NumberOfStates[j]);
        if (j == NumberOfFeatures - 1)
            sw.Write("{0,-6}", 1);
        else
            sw.Write("{0,-6}\t", 1);
    } else
    {
        sw_serial.WriteLine("{0} ",
        NumberOfStates[j]);
        if (j == NumberOfFeatures - 1)
            sw.Write("{0,-6}",
            NumberOfStates[j]);
        else
            sw.Write("{0,-6}\t",
            NumberOfStates[j]);
    }
}
sw.WriteLine();
}
sw_serial.Close();
sw.Close();
Directory.SetCurrentDirectory(currentDir);

// categorization for each feature
public static void categorization(int NumberOfFeatures, double[,] Range, int[] NumberOfStates, double[,] States, double[] IntermediateState)
{
    StreamReader statesFromTraining = new StreamReader("statesFromTraining.txt");
    for (int i = 0; i < NumberOfFeatures; i++)
    {
        NumberOfStates[i] = Convert.ToInt32(statesFromTraining.ReadLine());
        IntermediateState[i] = Range[i, 0];
        for (int j = 0; j < NumberOfStates[i] - 1; j++)
        {
            States[i, j, 0] = IntermediateState[i];
            States[i, j, 1] =
                .ToDouble(statesFromTraining.ReadLine());
            while (true)
            {
                if ((States[i, j, 1] >= States[i, j, 0]) &&
                    (States[i, j, 1] < Range[i, 1]))
                    break;
                else
                    States[i, j, 1] =
                        .ToDouble(statesFromTraining.ReadLine());
            }
            IntermediateState[i] = States[i, j, 1];
        }
    }
    statesFromTraining.Close();
}
/* BeliefComputation: Compute belief for the target feature */

* Author: Chaoli Cai
* Input: record_from_discretized.txt, subfolder
discrete_sample0_serial, discrete_sample1_serial, ...
* Output: detail info (anomaly/normal for each sample),
categorization, and statistical results
* Modified: 11/20/09
* */

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
namespace BeliefComputation
{
class BeliefComputation
{
    static void Main(string[] args)
    {
        // read info. (NumberOfSamples & NumberOfFeatures) from record_from_discretized.txt
        StreamReader sr_record = new StreamReader("record_from_discretized.txt");
        int NumberOfSamples = Convert.ToInt32(sr_record.ReadLine());
        int NumberOfFeatures = Convert.ToInt32(sr_record.ReadLine());

        // read info. from targetFeature.txt
        StreamReader tF = new StreamReader("targetFeature.txt");
        int targetFeature = Convert.ToInt32(tF.ReadLine());

        StreamReader PiRecord = new StreamReader("pi_record.txt");
        StreamWriter EvaluatePiRecord = new StreamWriter("EvaluatePi.txt");
        EvaluatePiRecord.WriteLine("Feature: " + targetFeature.ToString());
    }
}
}
StreamWriter EvaluateLambdaRecord = new
StreamWriter("EvaluateLambda.txt");
EvaluateLambdaRecord.WriteLine("Feature: " +
targetFeature.ToString());
StreamWriter EvaluateSigmaRecord = new
StreamWriter("EvaluateSigma.txt");
EvaluateSigmaRecord.WriteLine("Feature: " +
targetFeature.ToString());
StreamWriter EvaluateBeliefRecord = new
StreamWriter("EvaluateBelief.txt");
EvaluateBeliefRecord.WriteLine("Feature: " +
targetFeature.ToString());

int[] NumberOfStates = new int[NumberOfFeatures];
for (int i = 0; i < NumberOfFeatures; i++)
{
    NumberOfStates[i] =
        Convert.ToInt32(sr_record.ReadLine());
}

int MaxNumberOfStates =
    Convert.ToInt32(sr_record.ReadLine());

int NumberOfDatasets = 0;
StreamReader[] sr_file = new
StreamReader[NumberOfSamples];
int[] NumberOfRows = new int[NumberOfSamples];
int TotalCases = 0;

double[,] TempPi = new double[NumberOfFeatures, MaxNumberOfStates];
double[,] Total = new double[NumberOfFeatures, MaxNumberOfStates];
double[,] Mean = new double[NumberOfFeatures, MaxNumberOfStates];
double[,] Lambda = new double[NumberOfSamples, NumberOfFeatures, MaxNumberOfStates];
double[,] Belief = new double[NumberOfSamples, NumberOfFeatures, MaxNumberOfStates];
double[,] Pi = new double[NumberOfSamples, NumberOfFeatures, MaxNumberOfStates];
double[,] Sigma = new double[NumberOfSamples, NumberOfFeatures];
double[] SortBelief = new double[NumberOfSamples];

for (int r = 0; r < NumberOfFeatures; r++)
{
    for (int c = 0; c < MaxNumberOfStates; c++)
    {
        TempPi[r, c] = 0;
        Total[r, c] = 0;
        Mean[r, c] = 0;
    }
}

for (int d = 0; d < NumberOfSamples; d++)
{
    for (int r = 0; r < NumberOfFeatures; r++)
    {
        Sigma[d, r] = 0;
        for (int c = 0; c < MaxNumberOfStates; c++)
        {
            Pi[d, r, c] = 0;
            Lambda[d, r, c] = 0;
            Belief[d, r, c] = 0;
        }
    }
}

for (int i = 0; i < NumberOfFeatures; i++)
{
    if ((i + 1) == targetFeature)
    {
        for (int c = 0; c < NumberOfStates[i]; c++)
        {
            if (c == 0)
            {
                EvaluatePiRecord.Write("\t\tstate{0}" + (c + 1).ToString());
                EvaluateLambdaRecord.Write("\t\tstate{0}", (c + 1).ToString());
                EvaluateBeliefRecord.Write("\t\tstate{0}" + (c + 1).ToString());
            }

            // Additional code...
        }
    }
}
}`
else
{
    EvaluatePiRecord.Write("\tstate{0}",
    (c + 1).ToString());
    EvaluateLambdaRecord.Write("\tstate{0}",
    (c + 1).ToString());
    EvaluateBeliefRecord.Write("\tstate{
    0}", (c + 1).ToString());
}

EvaluatePiRecord.WriteLine();
EvaluateLambdaRecord.WriteLine();
EvaluateBeliefRecord.WriteLine();
}

// get testing samples from subfolder
for (int Sample = 0; Sample < NumberOfSamples; Sample++)
{
    if (File.Exists("discrete_sample" + Sample.ToString() +
    "_serial.txt"))
    {
        NumberOfDatasets++;
        sr_file[Sample] = new StreamReader("discrete_sample" +
    Sample.ToString() + "_serial.txt");
        NumberOfRows[Sample] =
    Convert.ToInt32(sr_file[Sample].ReadLine());
        TotalCases = TotalCases + NumberOfRows[Sample];
        int[,] CellValues = new int[NumberOfRows[Sample],
    NumberOfFeatures];
    for (int row = 0; row < NumberOfRows[Sample]; row++)
    {
        for (int col = 0; col < NumberOfFeatures; col++)
        {
            CellValues[row, col] =
    Convert.ToInt32(sr_file[Sample].ReadLine());
        }
    }
}
// compute causal reasoning
causalReasoningComputation(NumberOfFeatures,
MaxNumberOfStates, NumberOfRows, NumberOfStates,
CellValues, Sample, Pi, PiRecord, EvaluatePiRecord, TempPi,
Total, targetFeature);
}
else
continue;
}
PiRecord.Close();
EvaluatePiRecord.Close();

StreamReader sr = new StreamReader("mean_PLrecord.txt");
for (int r = 0; r < NumberOfFeatures; r++)
{
    if ((r + 1) == targetFeature)
    {
        for (int c = 0; c < NumberOfStates[r]; c++)
        Mean[r, c] = Convert.ToDouble(sr.ReadLine());
    }
}

// compute diagnostic reasoning
diagnosticReasoningComputation(NumberOfSamples,
NumberOfFeatures, NumberOfStates, Mean, Pi,
EvaluateLambdaRecord, Lambda, targetFeature);
EvaluateLambdaRecord.Close();

// compute sigma
sigmaComputation(NumberOfSamples, NumberOfFeatures,
NumberOfStates, Pi, Lambda, Sigma, EvaluateSigmaRecord,
targetFeature);
EvaluateSigmaRecord.Close();

// compute belief
beliefComputation(NumberOfSamples, NumberOfFeatures,
NumberOfStates, Pi, Lambda, Sigma, Belief, EvaluateBeliefRecord,
targetFeature);
EvaluateBeliefRecord.Close();

StreamReader sc = new StreamReader("structure_comparison.txt");
int[] same = new int[NumberOfSamples];
for (int i = 0; i < NumberOfSamples; i++)
    same[i] = Convert.ToInt32(sc.ReadLine());

double[,] min = new double[NumberOfFeatures, MaxNumberOfStates];
double[,] max = new double[NumberOfFeatures, MaxNumberOfStates];
StreamReader range = new
StreamReader("Training_range_serial.txt");

for (int r = 0; r < NumberOfFeatures; r++)
{
    if ((r + 1) == targetFeature)
    {
        for (int c = 0; c < NumberOfStates[r]; c++)
        {
            min[r, c] = Convert.ToDouble(range.ReadLine());
            max[r, c] = Convert.ToDouble(range.ReadLine());
        }
    }
}

StreamReader FN = new StreamReader("Filename.txt");
string ans = FN.ReadLine();

// apply decision rule for anomaly detection and write results to
"Cumulative_Results.txt"
StreamWriter SW;
SW = File.AppendText("Cumulative_Results.txt");

SW.WriteLine("-----------------------------------------------");
SW.WriteLine("-----------------------------------------------");

int[] res = new int[NumberOfFeatures];
int DiffStruc = 0;
for (int f = 0; f < NumberOfFeatures; f++)
    res[f] = 0;
for (int d = 0; d < NumberOfSamples; d++)
{
    bool foundAnomaly = false;
    if (same[d] != 1)
{ 
DiffStruc++;
if (ans == "FAR")
{
    Console.WriteLine("sample {0} is an anomaly!!!", (d + 1).ToString());
    SW.WriteLine("sample {0} is an anomaly!!!", (d + 1).ToString());
}
else if (ans == "DR")
{
    Console.WriteLine("sample {0} is normal!", (d + 1).ToString());
    SW.WriteLine("sample {0} is normal!", (d + 1).ToString());
}
continue;
}
else
{
    Console.Write("sample {0}:", (d + 1).ToString());
    SW.Write("sample {0}:", (d + 1).ToString());
    for (int r = 0; r < NumberOfFeatures; r++)
    {
        if ((r + 1) == targetFeature)
        {
            for (int c = 0; c < NumberOfStates[r]; c++)
            {
                if (ans == "DR")
                {
                    if ((Math.Round(Belief[d, r, c], 4) >= max[r, c]) || (Math.Round(Belief[d, r, c], 4) <= min[r, c]))
                    {
                        Console.WriteLine("{0} {1} {2}", Belief[d, r, c], max[r, c], min[r, c]);
                        foundAnomaly = true;
                        break;
                    }
                } else if (ans == "FAR")
                { 

                }
            }
        }
    }
}
else if (ans == "FAR")
{
if ((Math.Round(Belief[d, r, c], 4) > max[r, c]) || (Math.Round(Belief[d, r, c], 4) < min[r, c]))
{
    Console.WriteLine("{0} {1} {2}", Belief[d, r, c], max[r, c], min[r, c]);
    foundAnomaly = true;
    break;
}

}
if (ans == "FAR")
{
    Console.Write(" normal!", (d + 1).ToString());
    SW.Write(" normal!", (d + 1).ToString());
}
else if (ans == "DR")
{
    Console.Write(" anomaly!!!", (d + 1).ToString());
    SW.Write(" anomaly!!!", (d + 1).ToString());
}
}
}
Console.WriteLine();
SW.WriteLine();

StreamReader Title = new StreamReader("statesFromTraining.txt");

Console.WriteLine();
SW.WriteLine("-------------------------------");
for (int r = 0; r < NumberOfFeatures; r++)
{
    SW.Write("{0}: {1} [", (r + l).ToString(), Convert.ToInt32(Title.ReadLine()));
    for (int c = 0; c < NumberOfStates[r] - 1; c++)
        SW.Write("{0}; ", Title.ReadLine());
    SW.Write(']');
    SW.WriteLine();
}
SW.WriteLine("-------------------------------");
if (ans == "DR")
{
    SW.WriteLine("DR:");
    for (int r = 0; r < NumberOfFeatures; r++)
    {
        if (r == targetFeature - 1)
        {
            Console.WriteLine("DR of feature {0} is: 
            {1:f3}%", (r + 1).ToString(), ((double)(DiffStruc + res[r]) / NumberOfSamples) * 100);
else if (ans == "FAR")
{
    SW.WriteLine("\nFAR: ");
    for (int r = 0; r < NumberOfFeatures; r++)
    {
        if ((r + 1) == targetFeature)
        {
            Console.WriteLine("FAR of feature {0} is:
            {1:f3} \%", (r + 1).ToString(), ((double)(DiffStruc + res[r]) / NumberOfSamples) * 100);
            SW.WriteLine("\tf{0}: {1:f3} \%", (r + 1).ToString(), ((double)(DiffStruc + res[r]) / NumberOfSamples) * 100);
        }
    }
    SW.Close();
    FN.Close();
}

// compute causal reasoning (pi)
static void causalReasoningComputation(int NumberOfFeatures, int MaxNumberOfStates, int[] NumberOfRows, int[] NumberOfStates, int[,] CellValues, int Sample, double[,] Pi, StreamWriter PiRecord, StreamWriter EvaluatePiRecord, double[,] TempPi, double[,] Total, int targetFeature)
{
    Console.WriteLine("\nsample "+ Sample.ToString());
    int[,] freq = new int[NumberOfFeatures, MaxNumberOfStates];
    for (int f = 0; f < NumberOfFeatures; f++)
    {
        if ((f + 1) == targetFeature)
        {
            if (Sample < 9)
                EvaluatePiRecord.Write("\n\nsample "+ (Sample + 1).ToString() + "\t\t"};
else
    EvaluatePiRecord.Write("\nsample" + (Sample + 1).ToString() + "\t");
int k = 0;
for (int j = 0; j < NumberOfRows[Sample]; j++)
{
    for (k = 0; k < NumberOfStates[f]; k++)
    {
        if (CellValues[j, f] == k + 1)
            freq[f, k]++;
        break;
    }
}
Consent.WriteLine("\tfeature " + (f + 1).ToString());
for (k = 0; k < NumberOfStates[f]; k++)
{
    TempPi[f, k] = (double)freq[f, k] / NumberOfRows[Sample];
    Total[f, k] = Total[f, k] + TempPi[f, k];
    Console.WriteLine("\t\tstate " + (k + 1).ToString() + " pi: {0:f4} ", TempPi[f, k]);
    PiRecord.WriteLine("{0:f4}" TempPi[f, k]);
    EvaluatePiRecord.Write("{0:f4}\t", TempPi[f, k]);
    Pi[Sample, f, k] = TempPi[f, k];
}
EvaluatePiRecord.WriteUne();

// compute diagnostic reasoning (lambda)
static void diagnosticReasoningComputation(int NumberOfSamples, int NumberOfFeatures, int[] NumberOfStates, double[,] Mean, double[,] Pi, StreamWriter EvaluateLambdaRecord, double[,] Lambda, int targetFeature)
{
    for (int d = 0; d < NumberOfSamples; d++)
    {
        for (int r = 0; r < NumberOfFeatures; r++)
        {
if ((r + 1) == targetFeature)
{
    if (d < 9)
        EvaluateLambdaRecord.Write("\nsam
ple" + (d + 1).ToString() + "\t\t");
    else
        EvaluateLambdaRecord.Write("\nsam
ple" + (d + 1).ToString() + "\t");
    for (int c = 0; c < NumberOfStates[r]; c++)
    {
        if (Mean[r, c] == 0)
        {
            Lambda[d, r, c] = 1;
            EvaluateLambdaRecord.Write("\t", Lambda[d, r, c]);
        }
        else
        {
            Lambda[d, r, c] = Pi[d, r, c] / Mean[r, c];
            EvaluateLambdaRecord.Write("\t", Lambda[d, r, c]);
        }
        EvaluateLambdaRecord.WriteLine();
    }
}

// compute sigma
static void sigmaComputation(int NumberOfSamples, int NumberOfFeatures, int[] NumberOfStates, double[,] Pi, double[,] Lambda, double[,] Sigma, StreamWriter EvaluateSigmaRecord, int targetFeature)
{
    for (int d = 0; d < NumberOfSamples; d++)
    {
        for (int r = 0; r < NumberOfFeatures; r++)
        {
            if ((r + 1) == targetFeature)
static void beliefComputation(int NumberOfSamples, int NumberOfFeatures, int[] NumberOfStates, double[,] Pi, double[,] Lambda, double[,] Sigma, double[,] Belief, StreamWriter EvaluateBeliefRecord, int targetFeature)
{
    for (int d = 0; d < NumberOfSamples; d++)
    {
        for (int r = 0; r < NumberOfFeatures; r++)
        {
            if ((r + 1) == targetFeature)
            {
                if (d < 9)
                    EvaluateBeliefRecord.WriteLine("\nsample \{0:f4}\t\t", Belief[d, r, c]);
                else
                    EvaluateBeliefRecord.WriteLine("\nsample \{0:f4}\t", Sigma[d, r]);

            }
            Sigma[d, r] = Sigma[d, r] + Pi[d, r, c] * Lambda[d, r, c];
        }
    }
}

// compute belief (by applying the Bayes's rule)
static void beliefComputation(int NumberOfSamples, int NumberOfFeatures, int[] NumberOfStates, double[,] Pi, double[,] Lambda, double[,] Sigma, double[,] Belief, StreamWriter EvaluateBeliefRecord, int targetFeature)
{
    for (int d = 0; d < NumberOfSamples; d++)
    {
        for (int r = 0; r < NumberOfFeatures; r++)
        {
            if ((r + 1) == targetFeature)
            {
                if (d < 9)
                    EvaluateBeliefRecord.Write("\nsample " + (d + 1).ToString() + "\t\t");
                else
                    EvaluateBeliefRecord.Write("\nsample " + (d + 1).ToString() + "\t");

            }
            Sigma[d, r] = Sigma[d, r] + Pi[d, r, c] * Lambda[d, r, c];
        }
    }
}


[63] Jiawei Han and Micheline Kamber. *Data Mining: Concepts and Techniques*. Elsevier. 2006.


