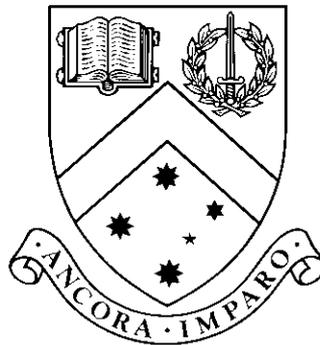


Distributed Associative Memory for Scalable Pattern Recognition within Peer-to-Peer Networks

by

Amiza Amir, Masters of Science



Thesis

Submitted by Amiza Amir

for fulfillment of the Requirements for the Degree of
Doctor of Philosophy (0190)

**Clayton School of Information Technology
Monash University**

September, 2014

© Copyright

by

Amiza Amir

2014

To my parents - Munah Salim and Amir Ali

Contents

List of Tables	x
List of Figures	xiii
List of Abbreviations	xvii
Abstract	xx
Acknowledgments	xxiv
Publications	xxv
1 Introduction	1
1.1 Background	2
1.2 Distributed Pattern Recognition within P2P Networks: Challenges	8
1.3 Motivation and Aims	10
1.4 Thesis Outline	13
2 P2P Networks and Distributed Pattern Recognition	15
2.1 P2P Networks	18
2.1.1 Characteristics of P2P Systems	18
2.1.2 P2P Network Overlay Architecture	20
2.1.3 Applications of P2P Networks	22
2.2 Current Recognition Practices in P2P Networks	26
2.2.1 Screening	26
2.2.2 Signature-based Approach	27

2.2.3	Reputation-based Approach	28
2.2.4	The Intelligent Pattern Classification Approach	29
2.3	Pattern Recognition	29
2.3.1	Eager and Lazy Learning	31
2.3.2	Supervised and Unsupervised Learning	32
2.3.3	Batch and Online Learning	32
2.4	Distributed Pattern Recognition	33
2.4.1	Distributed Pattern Recognition Components	33
2.4.2	Artificial Neural Networks	35
2.4.3	Ensemble Classifiers	39
2.4.4	Graph Neuron-based Algorithms	41
2.4.5	Network-wide Algorithms	42
2.5	Distributed Aggregation	48
2.5.1	Sequential-Reduction	49
2.5.2	Parallel-Multilayer Reduction	49
2.5.3	Aggregate-Broadcast	50
2.5.4	Flooding	50
2.5.5	Gossiping	51
2.5.6	Local Algorithm with Random Walk Sampling	51
2.5.7	Distributed Majority Voting	52
2.5.8	Distributed Plurality Voting	52
2.5.9	Best-effort Algorithm	53
2.6	Comparative Analysis	53
2.6.1	Comparison of Traditional Classification Approaches within P2P Networks	54
2.6.2	Task Distribution and Peer Involvement in DPR	55
2.6.3	Efficient Online Learning	57
2.6.4	Lack of synchronisation	60
2.6.5	Invariant to imbalanced data distribution	60

2.6.6	Scalability within Large Networks	62
2.6.7	Scalability for Large Datasets	63
2.6.8	Fault-tolerance	65
2.7	Summary	66
3	P2P-GN for Distributed PR within P2P Networks	71
3.1	Overview of P2P-GN	75
3.1.1	The Input of P2P-GN	75
3.1.2	Logical Structure of P2P-GN	78
3.1.3	Distributed Storage	80
3.1.4	Two Stage Prediction	81
3.1.5	Bias Element Identifier Generation	82
3.1.6	Network-wide P2P-GN	84
3.1.7	Learning Procedure	85
3.1.8	Recall Procedure	87
3.2	Experimental Results	93
3.2.1	Simulator and Classifiers Setting	94
3.2.2	Datasets	96
3.2.3	Comparative Accuracy Against Centralised Algorithms	99
3.2.4	Evaluating Efficiency for Large Networks	101
3.2.5	Evaluating The Effect of Imbalanced Data Distribution	106
3.2.6	Workload Distribution VS Network Size	109
3.3	Complexity Analysis	111
3.4	Summary	115
4	A P2P Classification Algorithm for Large Datasets	119
4.1	The Convergecast Recall for High-Dimensional Problems	123
4.1.1	Tree Construction	124
4.1.2	Convergecast Recall	127
4.2	Evaluating Communication Efficiency: Comparative Analysis	132

4.3	Evaluation Resource Efficiency for Large Datasets	138
4.3.1	The Datasets	139
4.3.2	Experiment 1: Evaluating Resource Efficiency on the Waveform- 40 and LED datasets	140
4.3.3	Experiment 2: Evaluating Time Efficiency on the MNIST dataset	145
4.4	Complexity Analysis	152
4.4.1	Storage Complexity Estimation	152
4.4.2	Local Computational Complexity At Peer Level	153
4.4.3	Overall Time Complexity At Network Level	155
4.5	P2P-GN within Dynamic Networks	158
4.6	Fault Management in P2P-GN	161
4.6.1	Fault Management Framework	162
4.6.2	Recovering From The Aftermath of Peer Joining	171
4.6.3	Recovering From The Aftermath of Peer Departure	173
4.7	Evaluating System Reliability	183
4.7.1	Simulation Set-Ups	183
4.7.2	Churn Modelling	184
4.7.3	Investigating The Churn Effect on the P2P-GN Performance .	185
4.7.4	Evaluating the Performance of Fault Tolerance Scheme	189
4.8	Summary	195
5	Distributed Spam Detection using DASMET	199
5.1	Efficient Classification for Datasets with Frequent Repeating Patterns	202
5.2	Distributed Spam Detection System	204
5.3	DASMET for P2P-based Pattern Recognition	206
5.3.1	The Logical Structure of The DASMET	206
5.3.2	Network-wide DASMET	210
5.3.3	Bias Identifier Generation	210
5.3.4	DASMET Learning Procedure	214

5.3.5	DASMET Recall Procedure	215
5.4	Experiment	217
5.4.1	Distributed Image Spam Problem	217
5.4.2	Distributed Email Spam Problem	218
5.4.3	Datasets	219
5.4.4	The Classifiers	222
5.4.5	The P2P-GN and DASMET Structure	224
5.4.6	Simulator Set-Up for Distributed Algorithms	225
5.4.7	Performance Metrics	225
5.4.8	Evaluating Accuracy	227
5.4.9	Evaluating Communication Overheads	231
5.4.10	Evaluating Recall Time	235
5.4.11	Evaluating DASMET Performance with Varying Parameters	240
5.5	Complexity Analysis	246
5.5.1	Message Complexity	246
5.5.2	Overall Time Complexity At Network Level	247
5.6	Summary	249
6	Conclusions and Future Work	253
6.1	Potential Applications	258
6.2	Limitations	260
6.3	Future Research	262
6.4	Summary	266
	Appendix A: Notations	269
A.1	Notations for Chapter 3	269
A.2	Notations for Chapter 4	271
A.3	Notations for Chapter 5	275
	Appendix B: P2P-GN	277
B.1	Generating Sub-patterns	277

B.2	An example of the P2P-GN recall process	279
Appendix C: Convergecast Recall		281
C.1	Maximum Simultaneous Connections Per Peer	281
C.2	Tree Construction	281
C.3	Discussion on Communication Overheads in the BPNN, HGN, DHGN and P2P-GN	283
C.4	Examples Architectures of the BPNN, HGN, DHGN and P2P-GN . . .	285
Appendix D: DASMET		289
D.5	Examples of DASMET Recall Procedure	289
D.6	Dataset Factor to The Probability of Optimal Recall Prediction . . .	290

List of Tables

2.1	Category of distributed PR algorithms based on task distribution . . .	56
2.2	Comparison on efficient online earning	58
2.3	Comparison on distributed pattern recognition approaches	68
3.1	Example of segments generated from $\mathbf{x} = \{high, high, 2, 4, medium, low\}$	77
3.2	Example of segments generated from $\mathbf{x} = \{high, high, 2, 4, medium, low\}$	77
3.3	Datasets for accuracy testing.	99
3.4	Accuracy comparison with centralised, state-of-the-art classifiers . . .	100
3.5	Accuracy test on the imbalanced class distribution scenario.	107
3.6	Accuracy test on decreasing sample sizes per peer (with upper bound of 300 to 50 samples per peer).	108
3.7	Summary of the communication complexity as a function of network size (N).	112
3.8	Summary of the recall time complexity as a function of network size (N).	115
4.1	The maximum number of messages per peer during learning and recall with varying d values.	134
4.2	Computational overheads of centralised state-of-the-art algorithms on the Waveform-40 and LED dataset.	143
4.3	Computational overheads of the P2P-GN on the Waveform-40 and LED dataset.	143
4.4	Local recall time, t_R (ms) in the flat recall with increasing stored data.	148

4.5	Local recall time, t_R (ms) in the convergecast recall with increasing stored data.	148
4.6	Local training time, t_L (ms) with increasing data dimension.	149
4.7	Local recall time, t_R (ms) during flat recall with increasing data dimension.	150
4.8	Local recall time, t_R (ms) during convergecast recall with increasing data dimension.	150
4.9	Hash table average performance	154
4.10	Local computational complexity (Big O) of the P2P-GN as a function of training dataset size (q) at the peer level	154
4.11	Local computational complexity (Big O) of the P2P-GN as a function of data dimension (d) at the peer level	155
4.12	Neighbourhood of peer h_{50} from the example in Figure 4.14. $p_{pre[i]}$ is an $(i + 1)$ -th predecessor of peer p where $p_{pre[0]}$ refers to its immediate predecessor. While, $p_{suc[i]}$ is an $(i + 1)$ -th successor of peer p where $p_{suc[0]}$ refers to its immediate successor.	164
4.13	The placement of replicas for bias element h_{41} from the example in Figure 4.16.	166
4.14	Events which trigger the recovering process following the peer p departure.	175
4.15	Input to the <code>routineRecovery</code> Protocol	178
4.16	Topology changes in a period 20,000 seconds using Weibull churn model with different $MTTF$	186
4.17	The accuracy with bias element availability under Weibull churn model with $MTTF = 5,000$ seconds at different points of simulation time.	188
4.18	The bias element availability and accuracy after $t_q = 7,000$ seconds under different $MTTF$	189
4.19	The recovery messages per second (M/t) within 20,000 seconds of simulation time.	194

4.20	The data transferred per second (L/t) within 20,000 seconds of simulation time.	194
5.1	The datasets used in this experiment	219
5.2	The P2P-GN and DASMET structures for this experiment.	225
5.3	Accuracy on the experiment using FCTH descriptor for image spam detection.	228
5.4	Accuracy on the experiment using CEDD descriptor for image spam detection.	228
5.5	Accuracy on the experiment using Haralick descriptor for image spam detection	229
5.6	Accuracy on the spam email dataset	231
5.7	Parameter d in all experiments	241
5.8	Parameter φ_s for FCTH, CEDD and email spam experiments	242
5.9	Summary of the DASMET complexity.	248
A.1	The notations for Chapter 3: P2P-GN For Distributed PR within P2P Networks	269
A.2	The notations for Chapter 4: A P2P Classification for Large Datasets	271
A.3	The notations for Chapter 5: Distributed Spam Detection	275
C.1	The Suggested Maximum Global Connection Per Peer	281
C.2	The architectures of three-layer BPNN for varying numbers of attributes d and the number of outputs $ \mathcal{C} = 10$	286
C.3	The architectures of the HGN and the DHGN with varying number of attributes d	286
C.4	The P2P-GN architectures with varying numbers of attributes.	287

List of Figures

1.1	Thesis outline	13
2.1	P2P applications are built on top of logical P2P overlay network within the application layer	20
2.2	Distributed pattern recognition components.	34
2.3	An example of BAMN architecture.	38
2.4	An example of GN structure for a binary pattern.	42
2.5	An example of HGN structure for a seven bits binary pattern	43
2.6	An example of local classification algorithms.	45
3.1	An example of logical P2P-GN structure view from a requester	78
3.2	An example of bias elements	81
3.3	The stored entry(s) in bias array(s) for five 5×7 -size binary images at seven leaf nodes.	81
3.4	An example of identifier generation by using pair of segment and position, or by using only segment as input.	83
3.5	An example of learning requests in the P2P-GN	88
3.6	An example of recall requests in the P2P-GN.	90
3.7	An example of responses in the P2P-GN	91
3.8	The two stage predictions during the final prediction process.	93
3.9	Some examples of handwritten images from the MNIST dataset.	97
3.10	\tilde{m} during recall on the dataset generated by the Waveform data gen- erator (Version 2).	103

3.11 \tilde{u} during recall on the dataset generated by the Waveform data generator (Version 2).	103
3.12 \tilde{m} in the P2P-GN with Chord and with optimal connection during recall on the dataset generated by the Waveform data generator (Version 2)	104
3.13 Average recall time per instance of the P2P-GN with Chord, Ivote-DPV and ensemble ID3 on the dataset generated by the Waveform data generator (Version 2)	105
3.14 Average recall time per instance of the P2P-GN with Chord on the dataset generated by the Waveform data generator (Version 2).	106
3.15 Workload (bias elements) distribution in the cluster-based implementation	110
3.16 Workload (bias elements) distribution in the network-wide implementation	111
4.1 The P2P-GN procedures involves a learning and two types of recall methods: the flat recall and the convergecast recall	121
4.2 Two examples of the convergecast recall tree structure for $d = 80$, $d_s = 5$, and $OV = 0$ (a) with undefined m , and (b) with $m = 8$	125
4.3 An example of a convergecast recall tree structure for $d = 80$, $d_s = 5$, $OV = 0$, and $m = 2$	126
4.4 Step 1: A spanning tree during convergecast recall	131
4.5 Step 2: The local predictions from leaf nodes are sent to the internal nodes	131
4.6 Step 3 and 4: The intermediate aggregation results from internal nodes are forward to the requester.	132
4.7 Number of messages with increasing data dimension during learning.	136
4.8 Number of messages with increasing data dimension during recall.	137
4.9 Average storage requirement S_μ in the P2P-GN.	144

4.10	Six peers which hosting leaf nodes by local learning time per instance (t_L) in the experiment on MNIST dataset.	147
4.11	Estimated overall run-time with an increase in data dimension.	157
4.12	The P2P-GN reliability through overlapping patterns	161
4.13	The fault management framework	163
4.14	An example of a peer's neighbourhood	164
4.15	The neighbourhood watch service.	165
4.16	Master and replicas placement	165
4.17	A sequence diagram for learning process.	167
4.18	The peer selection process	169
4.19	An example of the joining node causing an incorrect placement in the master copy location.	171
4.20	The loss of master copy after node departure	174
4.21	An example of insufficient number of replicas available within the system due to a peer departure.	175
4.22	The lifetime VS probability plot for the Weibull inverse survival function	184
4.23	The decreasing bias element availability under heavy lifetime churn .	187
4.24	MB_A for system recovery	192
4.25	RB_A for system recovery	193
4.26	The growth in network size (N) and $M_{routine}/P$ during routine recov- ery session	195
5.1	The key idea in the spam detection using DASMET	203
5.2	Functional components of the distributed spam detection system . . .	205
5.3	An example of a DASMET architecture	208
5.4	Examples of bias identifiers generation	211
5.5	The placement of bias elements within a Chord network.	213
5.6	Some examples of image spam	220
5.7	Some examples of image ham	221
5.8	Number of messages per test in the FCTH experiment	233

5.9	Number of messages per test in the CEDD experiment	234
5.10	Number of messages per test in the Haralick experiment	234
5.11	Number of messages per test in the email spam detection experiment	235
5.12	Recall time per test in the FCTH experiment.	236
5.13	Recall time per test in the CEDD experiment	237
5.14	Recall time per test in the Haralick experiment.	238
5.15	Recall time per test in the email detection experiment.	239
5.16	n_R with increasing φ_s	243
5.17	\hbar with increasing φ_s	244
5.18	Accuracy with increasing φ_s	245
B.1	An example of a P2P-GN recall process	279
D.1	An example of DASMET recall phase 1.	289
D.2	An example of DASMET recall phase 2.	290
D.3	An example of DASMET recall phase 3.	291

List of Abbreviations

***k*-NN** *k*-Nearest Neighbours

1-NN Nearest Neighbour

ADMM Alternating Direction Method of Multiplier

AI Artificial Intelligence

AM Associative Memory

BAM Bidirectional Associative Memory

BAMN Bidirectional Associative Memory Network

BPNN Backpropagation Neural Network

CAN Content Addressable Network

CD compact disk

CEDD Color and Edge Directivity Descriptor

CSVM Collaborative SVM

DHT Distributed Hash Table

DMV Distributed Majority Voting

DNS Domain Name System

DPV Distributed Plurality Voting

DPR Distributed Pattern Recognition

FCTH Fuzzy Color and Texture Histogram

GN Graph Neuron

GUID Global Unique Identifiers

HGN Hierarchical Graph Neuron

ID3 Iterative Dichotomiser 3

IM Instant Messaging

ISOLET Isolated Letter Speech Recognition

Ivote-DPV Distributed Ivote with Distributed Plurality Voting

LL2T Local L2-Thresholding

MAM Morphological Associative Memory

MB Megabyte

MFeat Multiple Features

MNIST Mixed National Institute of Standards and Technology

MoM-DSVM Method of Multiplier Distributed Support Vector Machine

MoM-NLDSVM Method of Multiplier Non-linear Distributed Support Vector
Machine

MTTF Mean-Time-to-Fail

ms milliseconds

NN Artificial Neural Network

OCR Optical Character Recognition

PAST Peer-to-Peer Archival Storage

PCM Pulse Code Modulated

PeDiT P2P Distributed Decision Tree Induction

P2P Peer-to-Peer

P2PTV P2P-based Internet television

P2P Bagging Cascade RSVM P2P Bagging Cascade Reduced Support Vector Machine

P2P Cascade RSVM P2P Cascade Reduced Support Vector Machine

PR Pattern Recognition

QoS Quality of Service

RBF Radial Basis Function

RBFN Radial Basis Function Network

RSVM Reduced Support Vector Machine

SVM Support Vector Machine

SOM Self Organising Map

TCP Transmission Control Protocol

TTL Time-To-Live

URL Uniform Resource Locator

WSN Wireless Sensor Networks

WWW World Wide Web

Distributed Associative Memory for Scalable Pattern Recognition within Peer-to-Peer Networks

Amiza Amir, Masters of Science
Amiza.Amir@infotech.monash.edu.au
Monash University, 2014

Abstract

Economical and scalability factors lead to a growing number of applications using P2P technology. However, the absence of a server to monitor the system makes these applications susceptible to uncontrolled distribution of undesired objects (e.g. pornography, polluted content, or spam). In centralised applications, this issue is solved effectively by using content filtering or spam filtering via pattern recognition. Hence, a similar solution is appealing for P2P-based applications. Within P2P networks, pattern recognition also has the potential to provide other classification services, such as music genre classification, attack detection and document's author recognition.

Pattern recognition in the P2P domain is resource intensive since involves large amounts of computation, significant complexity and rapid updating of data. The resource intensive computation typically requires a high-performance server which is unavailable in P2P networks. In order to deal with this limitation, a scalable, fully-distributed classification algorithm which performs fast processing is necessary. Thus, we consider an algorithmic perspective and explore design issues that arise in pattern recognition in P2P networks. Some of these issues include the absence of central coordination, low computational resources, lack of synchronisation, frequent data update, large dataset scalability, large network scalability, dynamic and imbalanced data distribution. None of the current P2P classification algorithms

deal with all of these issues, therefore, the proposed algorithm aims to adequately deal with all of these issues.

Considering that P2P networks are large and may have millions of peers, building an exact (not approximate) global classifier which represents the combined datasets across all peers is non-trivial. In current algorithms, every peer builds its own global classifier which results in several classifiers. These algorithms are fully-distributed and are guaranteed to converge to exact classifiers. This, however, is only possible through high communication overheads and processing times. In contrast, this thesis takes a different approach. It aims to show that exact classifications can be obtained at any time using limited communication by building a single in-network global classifier which is created through the collaboration among peers. This is achieved by devising an algorithm that consists of many loosely-coupled, atomic (fine-grained decompositions) storage units which are distributed across large networks. We refer to the resulting algorithm as the P2P-GN.

Existing methods perform a whole state-of-the-art classifier on a single peer. Although the local dataset at every peer is usually small, the computation is intractable due to the complexity of pattern recognition (particularly in dealing with high-dimensional problems). Additionally, the peers are reluctant to share their private resources inordinately. In contrast, in our method, each peer is only required to perform a simple pattern matching on a subset of features, and thus significantly reduces the computational overheads per peer. For high-dimensional problems, the aggregation is performed in parallel through the collaboration of several peers in a tree structure. This enables the algorithm to perform online learning to handle frequent local data updates and frequent prediction requests leading to processing large datasets efficiently. We also incorporate a fault tolerance scheme to preserve the system performance throughout the system lifetime within dynamic networks.

In order to assess the performance of the algorithm, we compare it to the existing centralised and distributed classification methods. Our experimental results on four datasets with a large number of attributes (i.e., Arcene, ISOLET, MNIST,

and MFeat) show that our method has comparable accuracy to the centralised classifiers. The efficiency of the algorithm for high-dimensional datasets is shown through a series of experiments on MNIST dataset. Additional experiments to validate the efficiency of the algorithm were performed on two large datasets (high number of samples) which were generated using two public data generators (Waveform (Version 2) and LED). The results also show that our method is invariant to imbalanced data distributions (imbalanced class distribution and small in size dataset per peer). We also analyse the algorithm’s scalability for large networks and its performance on a highly dynamic network. The results show that our algorithm incurs low communication overheads and the communication per peer is independent of the network size. This shows that the algorithm scales well, performing effectively irrespective of network size.

As a final example of the efficiency of the proposed algorithm we consider spam problems in P2P applications. We extend our algorithm to be an economical, scalable and fully-distributed spam detection system called the DASMET. This algorithm is specifically designed for datasets that consist of similar occurring patterns. We further demonstrate the application of DASMET for fully-distributed spam detection to two spam problems: email spam detection and image spam detection. The results show that the DASMET performs best with a relatively low amount of resources for the spam detection compared to other distributed methods. It is able to produce accurate predictions (99% accuracy for the email spam problem and 99% accuracy for the image spam problem by using Haralick descriptor) with minimal use of resources.

In conclusion, this thesis proposes the P2P-GN for efficient pattern recognition within P2P networks. The efficacy of the algorithm for datasets with large number of attributes and training samples are validated through a series of experiments. Furthermore, the scalability of the algorithm is demonstrated through experiments with varying network sizes. The experimental results show that the P2P-GN provides a practical alternative for scalable pattern recognition within P2P networks.

Declaration

In accordance with Monash University Doctorate Regulation 17 / Doctor of Philosophy and Master of Philosophy (MPhil) regulations the following declarations are made:

I hereby declare that this thesis contains no material which has been accepted for the award of any other degree or diploma at any university or equivalent institution and that, to the best of my knowledge and belief, this thesis contains no material previously published or written by another person, except where due reference is made in the text of the thesis.

Amiza Amir
September 17, 2014

Acknowledgments

Amiza Amir

Monash University

September 2014

Publications

Publications arising from this thesis include:

- Amir, A. and Raja Mahmood, R.M. and Khan. A. I. (2009)** Multi-wheel graph neuron: a distributed associative memory for structured P2P networks. In *Proceedings of 11th ACM International Conference on Information Integration and Web Based Applications and Services, (iiWAS2009)*, Kuala Lumpur, Malaysia, 14-16 December, 2009. pp. 147-154.
- Raja Mahmood, R. A, and Amin, A.H. M., Amir, A. and Khan, A. I.(2009)** Lightweight and distributed attack detection scheme in mobile ad hoc networks. In *The 7th International Conference on Advances in Mobile Computing and Multimedia (MoMM 2009)*, Kuala Lumpur, Malaysia, 14-16 December, 2009. pp. 162-169
- Amir, A. and Raja Mahmood, R.M. and Khan. A. I. (2010)** A Wheel Graph Structured Associative Memory for Single-Cycle Pattern Recognition within P2P Networks. In *Proceedings of 24th IEEE International Conference on Advanced Information Networking and Applications Workshops, (AINA 2010)* , Perth, Australia, 20-23, April 2010.
- Amir, A. and Amin, A.H. M. and Srinivasan, Bala (2011)** P2P-Based Image Recognition for Component Tracking in a Large Engineering Domain, in P. Ivnyi, B.H.V. Topping, (Editors), In *Proceedings of the Second International Conference on Parallel, Distributed, Grid and Cloud Computing for Engineering*, Corsica, France, 12-15 April, 2011, Civil-Comp Press, Paper 73.
- Raja Mahmood, R.A., and Amin, A. H. M, Amir, A., and Khan, A.I. (2012)** A lightweight graph-based pattern recognition scheme in mobile ad hoc networks, in *Trustworthy Ubiquitous Computing*, eds Ismail Khalil and Teddy Mantoro, Atlantis Press, Paris France, pp. 177-206.
- Amir, A. and Amin, A. H. M. and Khan, A. I. (2012)** A Multi-Feature Pattern Recognition for P2P-based System Using In-network Associative Memory. Technical Report 2011/265. Faculty of Information Technology, Monash University, Melbourne, Australia.
- Amir, A. and Amin, A. H. M. and Khan, A. I. (2013)** Developing Machine Intelligence within P2P Networks Using a Distributed Associative Memory. In *Algorithmic Probability and Friends. Bayesian Prediction and Artificial Intelligence*, Lecture Notes in Computer Science Vol. 7070, Springer Berlin Heidelberg, pp. 439-443.

Chapter 1

Introduction

P2P technology enables a large distributed community of computers to pool their resources to benefit each other. The widespread utilisation of P2P technology and its popularity have proven its efficiency and effectiveness as a direct sharing tool for multimedia files (e.g., text files, images, audio, video) and software distribution. Unfortunately, due to its gaining popularity, it faces challenges in dealing with the problem of spam (e.g., polluted files, polluted chunks in multimedia streaming applications, email spam, etc.). The popularity of P2P technology attracts businesses to promote their services or products through P2P applications. This is because it is an economical way of advertising yet enables them to reach a large audience easily. This causes a large amount of spam to be injected into P2P applications everyday. The affected applications including messenger applications such as Yahoo Messenger and Skype, media streaming and file-sharing applications.

Pattern recognition techniques have been proven effective for centralised spam detections [116, 161, 127]. Nonetheless, the decentralised nature of P2P networks warrants a fully-distributed approach of such algorithms. Pattern recognition is concerned with observing the environment, distinguishing patterns of interests and making decisions based on the observed patterns. It has been successfully applied to a number of domains such as music classification [45], handwritten recognition [200], image recognition [173] and human action recognition [188]. This shows that pattern recognition is not only useful for spam filtering within P2P networks but may also

benefit the available P2P-based applications in providing various services to improve their user interface. In addition, the fully-distributed, P2P-based pattern recognition schemes provide an economical alternative for large-scale pattern recognition—such large-scale pattern recognition for complex problems requires expensive infrastructure in centralised or grid-based system.

In the context of P2P networks, several pattern recognition algorithms have been proposed [125, 7, 17, 108]. However, there are still several limitations. In particular, the current P2P-based pattern recognition approaches incur high communication overheads, large processing times and high computational effort to obtain an *exact global classification*^{1.1}. This thesis proposes an algorithm which can provide exact global classifications, reduces resource usage and improves run-times, while maintaining scalability of the algorithm within large distributed networks. Considering that the amount of globally-shared data over networks is large, relatively few studies have been conducted to answer questions such as how to incorporate pattern recognition when dealing with complex and large data within a P2P environment.

1.1 Background

Many studies have been conducted on distributed machine learning, which can be classified into two categories: *cluster-based algorithms* and *network-wide algorithms*. Cluster-based algorithms are well-structured into a fixed architecture and are usually designed to improve accuracy, speed up computation, and process large datasets. Network-wide algorithms, on the other hand, are unstructured system with dynamic architecture that expand and shrink depending on the network and are used for large distributed networks.

A few examples of well-known cluster-based algorithms are artificial neural networks [185, 16, 137], ensemble classifiers [23, 169, 209], and algorithms that are built using the MapReduce framework [55, 41]. Ensemble classifiers such as bagging [23],

^{1.1}An exact global classification equals to the centralised classification which is obtained when the whole data within a network is located at a single site.

boosting [169], and stacking [209] build a number of weak classifiers on partition of the dataset, and then finally construct a better meta-classifier by combining and manipulating outcome from these weak classifiers. Caragea et al. [32], Nasution and Khan [140], Cheng et al. [38] and Muhamad Amin and Khan [138] show that communication overheads can be reduced by centralising only parts of the computation. Nonetheless, these cluster-based algorithms are mostly intractable to implement within real-world P2P systems, given that the systems lack servers, are asynchronous, are large and distributed, and are dynamic.

However, very few studies have been conducted to investigate an efficient network-wide distributed algorithm within the P2P set-up. These algorithms include P2P Distributed Decision Tree Induction (PeDiT) [17], Ivote with Distributed Plurality Voting (Ivote-DPV) [125], P2P Cascade Reduced Support Vector Machine (P2P Cascade RSVM) [7], P2P Bagging Cascade Reduced Support Vector Machine (P2P Bagging Cascade RSVM) [8], distributed boosting [108], and a group of distributed Support Vector Machine (SVM) (i.e., Method of Multiplier Distributed Support Vector Machine (MoM-DSVM), online MoM-DSVM and Method of Multiplier Non-linear Distributed Support Vector Machine (MoM-NLDSVM) [66]).

A distributed pattern recognition algorithm may either perform *aggregation during learning* to produce a global classifier or perform *aggregation during prediction* to produce a global prediction. Given a network G_P , a peer p_i , has a dataset b_i and $\forall p_i \in G_P$. Network-wide classifier algorithms aim to produce either *approximate global classifiers/predictions* or *exact global classifier/predictions*. The approximate global classifiers/predictions are made by using datasets from a subset of peers in G_P . The exact global classifiers/predictions, on the other hand, represent all datasets ($\bigcup_{i \in G_P} b_i$) within G_P . Building an exact global classifier/prediction is difficult in a serverless system, particularly when dealing with a highly distributed data within a very large network.

In the P2P Cascade RSVM and distributed boosting, every peer builds their own meta-classifier by combining local models from all other peers. As they consider all sites in building the meta-classifiers, the resultant classifiers represent the whole dataset across the network. However, this comes at a cost where the communication associated with each model update is quadratic in the network size. Thus, they do not scale well within large networks. To solve this problem, most of the available algorithms build their classifiers or predictions using approximate solutions where the decisions are made based on the best assumption they have about the global datasets—producing approximate global classifiers/predictions. This can be achieved by sampling or *local algorithms*^{1,2}.

The Collaborative SVM (CSVM) [145] and the P2P Bagging Cascade RSVM are two examples of algorithms which uses sampling approach for this purpose. They require low communication overheads compared to other methods since they aggregate solutions from only a small number of peers without any iterations. Nonetheless, the classifications of both algorithms—that are approximate global classifiers/predictions—are non-deterministic, since changes of neighbours in the CSVM or randomly selected peers for aggregation in the P2P bagging cascade RSVM may change the outcome. They can also be inaccurate as they only involve a small portion of the dataset.

Two examples of local algorithms for classification are the PeDiT and Ivote-DPV. In these algorithms, peers only communicate with a small number of peers and therefore, it is scalable for large networks. Local algorithms are guaranteed to converge to exact global classifications (building exact global classifiers/predictions); at a cost of an expensive communication overhead which grows linearly in network size [17]. PeDiT uses the Distributed Majority Voting (DMV) [207], while the Ivote-DPV uses Distributed Plurality Voting (DPV). Both DPV and DMV are *reactive*^{1,3}—a peer is only active due to the changes of neighbourhood or significant changes to local

^{1,2}A local algorithm is a distributed algorithm that has constant run-time, independent of the size of the network.

^{1,3}A reactive algorithm is only executed upon several predefined conditions.

statistics. In both schemes, an active peer sends voting instances to its immediate neighbours. In the case of PeDiT, where the number of majority voting instances increase exponentially with the number of distinct attribute values. Hence, this significantly increases the communication overheads for high-dimensional datasets.

Forero et al. [66] discusses three fully-distributed SVM-based local algorithms: MoM-DSVM, online MoM-DSVM and MoM-NLDSVM. Online MoM-DSVM is an online version of the MoM-DSVM while the MoM-NLDSVM solves non-linear problems. Similar to the PeDiT and the Ivote-DPV, these algorithms have convergence issues for all sites to reach a consensus. Furthermore, communication is expensive with high bandwidth requirements where peers must broadcast their estimates to all neighbours frequently.

Most of these algorithms assume their local classifiers have sufficient examples and have uniform class distribution at every site, which cannot be guaranteed in the real world P2P environment [9, 108, 8, 125]. Since every site collects its own local data, the imbalanced data distribution can lead to incorrect local solutions [9, 23, 14, 153, 65, 216]. The *imbalanced data distribution* in this thesis is referred to an imbalanced class distribution and small local datasets. The imbalanced data distribution can eventually lead to an incorrect global solution or long time for convergence to consensus.

In summary, every local classifier in network-wide algorithms perform computations that involves all attributes, which may result in expensive local computations particularly when dealing with high-dimensional data. Moreover, the frequent data update within P2P systems results in frequent model rebuilding, and this increases the local computational overheads. This not only discourages the peer participation, but may also limit the system scalability in processing large dataset. Therefore, the constraint on the use of resources, particularly within the network involving mobile devices—such as smart phones or tablets—will be considered in this thesis.^{1.4}

^{1.4}Not all users from modern desktop computers are willing to contribute a significant portion of their available resources, since they are not fully-dedicated to a specific computationally-intensive task.

Another issue is that the connection in P2P technology does not guarantee a fast interconnection between peers. Hence, the iterations in local algorithms (e.g., PeDiT and Ivote-DPV) results in a significant delay of response to learning or classification requests—this is not feasible for *online learning*. The P2P Cascade RSVM and the distributed boosting aggregates local solutions from every peers and so the computation time increases linearly with an increase in network size. The CSVM and the P2P bagging cascade RSVM may give a fast response, however, they unable to produce exact global classifications. Minimal iterations and parallel connections are helpful to speed up the in-network computation, given the high potential of frequent learning and classification request within large P2P networks.

An associative memory model is a class of neural networks, which is readily distributed with high parallelism and can be a potential alternative to build a fast P2P-based pattern recognition algorithm. It is able to recall a memory through the associations between distributed stored patterns and it has been widely used for pattern recognition in various domain such as in image recognition [46, 219], face recognition [217] and abstract object recognition [194]. This technique is robust as it can make corrections if there is a fault or missing information in the input pattern. Given the distributed memory approach of the associative memory method, it is naturally suited to the physically distributed memory in P2P set-up. The issues with most of the available associative memory algorithms (e.g. Hopfield network, Bidirectional Associative Memory (BAM), and Morphological Associative Memory (MAM)) are that they are highly iterative and consist of tightly-coupled processes.

The Graph Neuron (GN) [99] and Hierarchical Graph Neuron (HGN) [140] are single-cycle associative memory algorithms with simple computation. The low communication overheads and low computational requirement of these algorithms suit the heterogeneous P2P environment^{1.5}. Nonetheless, the HGN and GN have tightly-coupled processes and require a strict synchronisation in their computation, making it, unsuitable for asynchronous P2P networks. In addition, the available associative

^{1.5}A peer's performance can range from a high-performance modern desktop, to a resource-constrained tablet PC or smartphone.

memory models have a fixed structure that is not scalable for large distributed networks. The mapping of these models on P2P overlay requires the formation of peer cluster within a network. This is infeasible for large networks as this cluster can be a bottleneck to serve all peers. The HGN improves with the accuracy of the GN by propagating the results in a hierarchical structure and performs parts of the computation at a fusion centre (server-like host). The Distributed HGN (DHGN) [138] reduces the communication in the HGN. However, it remains semi-distributed and requires a significant magnitude of communication compared to the GN. In this thesis, we show how the GN structure can also be accurate with less communication cost, compared to the HGN and DHGN by centralising part of the computation at a fusion centre. Thus, synchronisation can be avoided by having joint features at nodes, instead of a node needing an input from its left and right neighbours. Then, we show that by using a simple hybrid majority voting and weighting approach, an accurate classification can be made at any peer, which and subsequently eliminates the need of any server-like host—working in a fully-distributed structure.

The new fully-distributed algorithm, however, is inherently a cluster-based type of algorithm. In cluster-based algorithms, the growing network increases the workload to a group of peers (a subset of peers within the network). This results in these peers become bottlenecks. Our next strategy in this research is to build a single in-network global classifier which represents the whole global dataset and distribute the computation across the network as much as possible. This gives us another reason to select the HGN and GN as a basis of this new algorithm, since they are distributable into fine-granularity components. These components are then managed across a dynamic and large network by using the Distributed Hash Table (DHT) scheme, which also provides an efficient linking of these components. The resultant dynamic algorithm also preserves the *locality*^{1.6} aspect, hence it is scalable for large distributed networks.

^{1.6}The ability of a peer to make a decision with the helps of a constant number of other peers.

The peers in P2P networks are usually dynamic and unpredictable [181]. This can significantly impact the system regarding data correctness and availability, and may disrupt the routing mechanism, which causes delay in linking peers. Most of the available algorithms for P2P classification are able to operate within a dynamic network, and lack asynchronism to deal with possible link delay [125, 108, 8, 17]. The distributed nature of the proposed algorithm provides a degree of fault-tolerance to the system. However, some fault tolerance scheme are still required to ensure the all the required information remains in the dynamic system under continuously heavy churns in order to preserve the system correctness to its steady state^{1.7}. Therefore, we investigate an efficient fault tolerance mechanism for our approach by devising a reactive fault tolerance scheme which operates under restrictive conditions to limit the communication during recovery process. In addition, the use of fine-grained storage components in our proposed classification algorithm helps to maintain the small overheads for recovery.

1.2 Distributed Pattern Recognition within P2P Networks: Challenges

The characteristics of P2P networks result in the following situations for distributed pattern recognition, which require attention in designing distributed pattern recognition algorithms within P2P networks.

- **Large network size** - The scalability of P2P networks results in large networks comprising hundred thousands to millions of peers. In computing a global classifier or prediction—which represents the overall knowledge across the network—every site needs to be included. Hence, the aggregation method used must be scalable for large networks.
- **Large total data** - In a large distributed system, every site collects their own data, and this results in large accumulative data. It is difficult to compute

^{1.7}The ability to perform correctly and efficiently as operating in a network without churn.

PR algorithms on such large dataset in the absence of a high-performance machine.

- **Frequent data update** - The high magnitude of sites also leads to a frequent data update or introduction of new data into the system—which requires a fast and efficient learning algorithm.
- **Imbalanced data distribution** - The imbalanced data distribution— imbalanced class distribution and small local datasets—can result in incorrect local models or local predictions [23, 14, 153, 65, 216].
- **Asynchronous network** - The links among nodes have no guarantee of a fast connection within the P2P systems, as peers are mainly connected through the Internet. Therefore, the distributed system should perform relax synchronisation in its computation.
- **Heterogeneous network** - Modern users use P2P technology not only from desktop computers, but also from mobile devices like smartphones and tablet PCs that depend on battery power and in average have less processing power than desktop computers. The problem is that the available state-of-the-art PR methods (which are usually used as local classifiers in the available network-wide classification algorithms) are usually involve a computationally expensive task. Unlike machines in grid computing which are usually dedicated for a specific task, machines in P2P systems are not expected to dedicate a large portion of their resources for any specific application. Therefore, for a distributed PR application to be practical for use within P2P systems as well as to encourage peer participation, it must consider the resource-awareness factor.
- **Dynamic network** - The distributed computing approach in P2P systems makes them less prone to a single point of failure (as faced by client-server systems), however, they are highly dynamic. A distributed implementation of PR within these networks may suffer from loss of data if no proper mechanism is applied to maintain the stored data.

1.3 Motivation and Aims

The classification and detection tasks—particularly in large and complex domains—are easier with the aid of intelligent systems. However, the modern classification methods using computational intelligence have yet to benefit the P2P community. Two possible reasons that hinder the adoption of pattern recognition within P2P networks are the complexity of the available distributed algorithms, and the high computational overheads which may discourage peer participation. In the rapidly expanding number of mobile devices (e.g. smartphones and tablet PCs) connected to the internet, alongside with modern desktop computers, a large number of peers in the P2P networks today can be these mobile devices [82]. The proposal of a simple algorithm with highly parallel processes, and with awareness on the computation overhead per peer in this thesis is appealing to benefit the future P2P community.

P2P distributed approaches have gained attention in machine learning research in dealing with the problem of geographically-distributed data. Given the large amount of collective knowledge within the network, manipulating this knowledge may lead to better results. The geographically-distributed nature of data restricts access to the whole training data for optimal recognition. The most common method is to gather all the raw data at a single central site where the data mining process can be performed in a conservative way. This requires a powerful centralised server—which is expensive—to store and perform computations on large combined dataset. The fast growing magnitude of data shared over the Internet may requires this infrastructure to be frequently upgraded, which is very costly. P2P-based systems, on the contrary, are economical and scale well with large data.

Another downside of a centralised approach is that it is prone to a single point of failure where the collapse of the server will completely collapse the whole system. Frequent data updates from many peers within the network may also result in a bottleneck problem. Compared to the centralised system, a distributed system eliminates the bottleneck issue and minimises the risk of total system failure,

at the cost of small degradation of system performance, and extra system maintenance. Also, the high number of peers in the P2P system enables replication, which improves the system's fault-tolerance [143].

P2P technology is mainly used for applications that involve rich and complex data e.g., audio files, videos, and images [117, 72]. The classification task involving these complex data usually deals with high-dimensional data, which limits the ability of the available classification algorithms to analyse the problem [79, 151, 93]. The effective solutions to this problem are mainly feature reduction techniques that require a computationally-intensive process in analysing the data [121, 204, 112, 12]. Hence, this inspire us to study an alternative solution to improve the scalability of our algorithm for high-dimensional problems.

The appealing benefits of distributed P2P-based systems and the challenges to build a scalable classification algorithm within the P2P networks attract us in pursuing this study. The main objective of this thesis is to identify an exact distributed classification algorithm that is scalable for large distributed systems, has a fast learning time, and is able to process ad hoc queries with rapid execution time. An exact distributed classification is an algorithm which can produce the same solution as the centralised implementation—that is when all data are available at the same location. To achieve this, we identify an available structured distributed algorithm with high parallelism which can be distributed into fine-granularity components, that is GN. These components are then distributed across P2P networks and the linking to these components are performed efficiently by using Distributed Hash Table (DHT) system. We show that this new approach is more efficient—in terms of run time and communication overhead—than the available distributed algorithms for large distributed networks.

Our work is different compared to other P2P classification algorithms where instead of building several global classifiers locally, we build a single global classifier within the network. We also focus on approaches that perform aggregation during

prediction^{1.8} to combine the local predictions, given that the rapid data update requires a fast online learning scheme. To our knowledge, the only well-known P2P classification algorithm that performs such way is the Ivote-DPV; hence, it is mainly used as a benchmark for the performance of the proposed system in this thesis. The work in this thesis aim to validate the following hypotheses:

- The proposed distributed algorithm that performs exact global classification, the P2P-GN, is scalable for large distributed networks and incurs small run-time and communication overheads.
 - The modified GN algorithm with hybrid weighting and voting approach is online, fully-distributed and asynchronous. The accuracy of this algorithm is comparable to the centralised, state-of-the-art algorithms.
 - The proposed approach has a low communication overhead and fast classification run-time. The communication overhead per peer is independent of the network size. Hence, the P2P-GN is scalable for large-scale distributed networks.
 - The proposed algorithm is invariant to an imbalanced data distribution.
 - The fine-granularity distribution of the proposed algorithm’s computation by using DHT system provides load-balancing of the system that is scalable for large networks.
- The proposed algorithm is scalable for large datasets.
 - The proposed approach provides an economical option for scalable classification algorithm of large datasets through a fully-distributed approach. The peer level (local) computational overhead (i.e., run-time, storage and communication) remains low in processing large datasets.
 - The online learning approach is time-efficient for classification of large datasets.

^{1.8}The aggregation is performed during the prediction phase when the local prediction results from all sites are combined into a global prediction result.

- The convergecast aggregation method improves the scalability for classification of high-dimensional dataset.
- The proposed algorithm is able to provide accurate predictions at any time throughout system lifetime despite operating within highly dynamic networks, with the support from an efficient fault-tolerance scheme.
- The P2P-GN performance when dealing with datasets with a high number of exact duplicates and near duplicates instances (e.g., spam detection), can be improved by the proposed extension of the P2P-GN, called the DASMET.

1.4 Thesis Outline

This dissertation is organised as follows (see Figure 1.1). In this present chapter

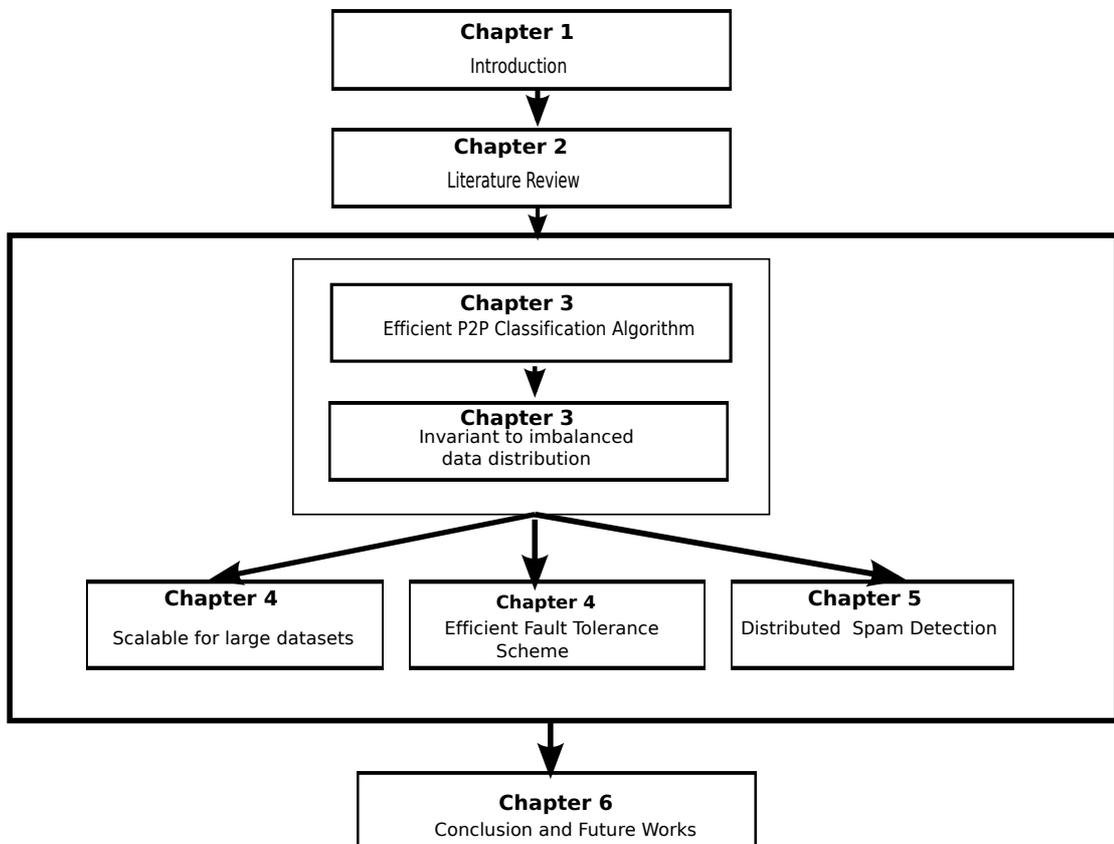


Figure 1.1: Thesis outline

(Chapter 1), the motivations and aims of this thesis are introduced. This followed by the organization of the dissertation. Chapter 2 presents a foundation and background of our research. It discusses different types of P2P networks and applications. In this chapter, we also discuss some potential applications of pattern recognition within the P2P domain. We then discuss the existing algorithms for distributed pattern recognition and particularly P2P-based pattern recognition algorithms. Next, the chapter summarises factors to build a distributed pattern recognition algorithm which is able to work effectively and efficiently within P2P networks; followed by the comparison between the existing algorithms against these factors.

Chapter 3 presents an online, asynchronous, and fully-distributed P2P-based in-network classification algorithm for large-scale distributed networks. The accuracy is compared to the state-of-the-art algorithms and the scalability of this approach is evaluated on a large-scale P2P networks. Then, we also demonstrate the invariant of our proposed algorithm in dealing with the imbalanced data distribution.

In Chapter 4, this thesis demonstrates scalability of this approach for large datasets and we show its scalability for high-dimensional problems. We also present an efficient fault tolerance scheme for the proposed algorithm in dealing with dynamic environment. This fault tolerance scheme is then evaluated against its communication overheads and its ability to preserve the data availability.

In Chapter 5, this thesis introduces a logical tree-structured associative memory algorithm that improves the algorithm which has been presented previously in Chapter 3 for datasets which have a high number of duplicates and near-duplicate data. This chapter particularly focuses on spam detection problems. The improvements includes the accuracy, the classification time and communication overheads. Finally in Chapter 6, we conclude the dissertation and present some prospective areas for the future research.

Chapter 2

P2P Networks and Distributed Pattern Recognition

P2P networks provide an economical option for building a serverless and scalable system. Thus, there is an increasing wide range of applications on P2P platforms which warrants a pattern recognition to improve these applications. Classification within P2P networks is usually performed using traditional approaches such as screening, reputation systems, and signature blacklisting. These traditional approaches seem to be more practical to implement within P2P environment compared to intelligent approaches; as the available pattern recognition algorithms^{2.1} are computationally-intensive and have tightly-coupled computations, which require a single processing site. While most researches are focussing on building a highly accurate and sophisticated pattern recognition algorithm, this hardly benefits systems with distributed data and processing. Accordingly, several distributed algorithms have been proposed from the literature to accommodate the need of pattern recognition for distributed environments [17, 7, 66, 8].

The two most popular reasons to study a distributed pattern recognition algorithm are to speed up the computation and to improve the classification accuracy. Firstly, distributed approaches are commonly used to optimise the computation speed through a divide-and-conquer approach by breaking a classifier into several

^{2.1}Pattern recognition is an intelligent tool for classification.

different small components, which are then assigned to several parallel processing units [140, 183]. Secondly, ensemble learning or a multi-classifier approach is used to iteratively build a more accurate classifier from many classifiers [34, 169, 23, 209]. In these approaches, multiple training sets are created (one training set per classifier) using different kinds of sampling methods. These training sets are placed distributively at multiple sites within the network and then a multi-site processing approach is used where each classifier learns its local training dataset independently. A highly accurate classifier is built iteratively by getting a consensus of results for the combination of many different classifiers.

Other than these two reasons, two more reasons are gaining popularity in studying distributed pattern recognition algorithms. Firstly, there is a new group of distributed pattern recognition algorithms which focus on building a global classifier from a large network. Given a large P2P network where every peer has its own dataset, a naive approach to perform pattern recognition is to gather all datasets from all peers at a single site and to execute the pattern recognition computation at the site. However, this approach is obviously inefficient as it requires a large magnitude of bandwidth consumption, a large number of communications and a high computational complexity at a single site. Without the existence of a high-performance peer, this approach is intractable considering the large number of peers within P2P networks and a large volume of combined data from these peers. Solving this issue requires a distributed approach which scalable for large networks. Several works on designing such algorithms have been proposed (e.g. PEDiT in [17], P2P Cascade RSVM in [7] and MoM-NLDSVM in [66]).

Secondly, the distributed approach can be used as a method to improve the pattern recognition scalability in dealing with large datasets. Most applications these days involve large and complex data. The data explosion phenomena (also known as Big Data) which is experienced by most organisations requires significant effort

to store, manage, access and protect the sensitive data [129, 130, 3]. While infrastructures in most organisations are unable to cope with the data explosion^{2,2}, the required containment cost (e.g., upgrades, procurements and maintenance costs) to increase software and hardware capability to handle the large dataset is significantly high. This restricts the system's ability to analyse and perform pattern recognition algorithms on this large and complex data.

The available P2P classification algorithms [17, 7, 66, 8] focus on improving the algorithm's scalability for large networks. They often neglect to improve the algorithm's scalability with respect to large datasets or to optimise the algorithm's run-time. In contrast, we investigate the potential of P2P networks to provide a scalable algorithm for large datasets, i.e., datasets with a high number of training datasets and a large number of attributes/features. P2P technology offers an economical way to handle large datasets by storing and analysing parts of the dataset separately at multiple distributed sites, and the local predictions or local classifiers (which are built at every site) are combined through aggregation. The ability to classify larger training datasets and high multi-dimensional may also improve the classification accuracy. Several strategies are involved in achieving this goal such as devising a highly distributed, online learning and efficient (time, storage and communication) algorithm. Considering that P2P networks are dynamic and unpredictable, it is challenging to achieve accurate predictions. Moreover, unlike multi-core or grid computing, the connections between nodes within P2P networks have greater latency; hence synchronisation is non-trivial.

Starting with a general overview of the P2P networks in Section 2.1, this chapter reviews the characteristics of P2P networks. The efficiency of a distributed pattern recognition algorithm relies on the connection between nodes, hence we examine the efficiency of the different architectures of P2P networks. Then, we discuss some examples of applications [71, 213, 177, 205, 18, 96, 118] that are

^{2,2}It was reported that data capacity on average in enterprises is growing at 40% to 60% per year [130]

built on P2P platforms to show the wide range of applications which may benefit from this research. The current practice of recognition and classification in P2P networks [113, 90, 198, 94, 44, 92] are discussed in Section 2.2. This is followed by an overview of pattern recognition in Section 2.3. Then, we review on the related work that has been done towards having distributed pattern recognition algorithms [125, 7, 17, 66], to examine the weakness and strength of the available approaches in achieving our research goal in Section 2.4. Distributed aggregation is one of the key components in a distributed pattern recognition algorithm which affects its effectiveness and efficiency. Therefore, in Section 2.5, we review several distributed aggregation approaches which have been used for combining local results, classifiers and statistics in distributed algorithms. In Section 2.6, we examine and perform comparison of the available classification approaches within P2P networks. In the summary in Section 2.7, we discuss issues arising in the current implementations and issues we are dealing with in implementing a scalable distributed pattern recognition algorithm within P2P networks.

2.1 P2P Networks

We provide an overview of P2P systems encompassing their characteristics and architectures. A review of several well-known applications which are built on top of P2P platforms are discussed. This also shows the wide range of P2P-based applications that may benefit from this work.

2.1.1 Characteristics of P2P Systems

Unlike client-server networks, where network information is stored on a centralised file server and made available to hundreds, thousands, or millions of client PCs, the information stored across P2P networks is distributed. The characteristics of P2P systems are as follows:

-
- (a) Equal roles: Each connecting and communicating peer is equal to one another and there are no ‘servers’ available unlike the client server model.
 - (b) Network scalability: Most P2P networks are highly scalable and may comprise hundreds of thousands or even millions of peers.
 - (c) Cost-efficient: The peers are usually the common users’ PCs which are not dedicated for a specific job. By sharing the storage and capability with each other, the total cost of ownership is lower than that of a central server. The network is efficient (fast and cheap) to set up and maintain as there is no need for a network administrator.
 - (d) Self-organising: There is no centralised node to control the peers in the network. Thus each node must be able to handle the joining, leaving and failure of nodes locally. The security solutions and data maintenance must be performed in a decentralised way without any single authority to handle the whole network as in the client-server system.
 - (e) Dynamic: Without a well-defined policy as in the grid-based system, the network membership is dynamic where nodes can leave and join the network at anytime. This improves network scalability, however this results in network churn causing an unstable and chaotic network.
 - (f) Reliability: Each peer can make backup copies of its data to give other peers facility for fault tolerance purposes. If any of the peers fail, the data is often still available at another peer. With the decentralised nature of P2P networks, they are less vulnerable to single point of failure compared to the centralised approach that depends on the server.
 - (g) Direct resource sharing: Peers can share disk space or files with each other directly without the mediation of a centralised server.

P2P technology allows PCs all over the world to connect with each other through direct internet connections even when servers are down. However, a system operation on a P2P network is more complex than on client-server architecture as there is no single entity with full control of the systems. Thus, the systems may face unanticipated behaviours. The appealing characteristics of the P2P networks outlined above have attracted researchers to study and use these networks more widely.

Figure 2.1 shows the P2P systems within the application layer. P2P overlay networks are logical networks which are built within the application layer and P2P applications are built on top of the logical network overlays.

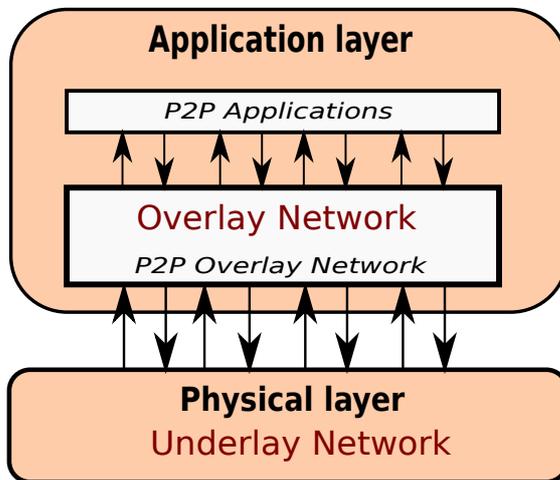


Figure 2.1: P2P applications are built on top of logical P2P overlay network within the application layer. The overlay network is built on top of the underlay network within the physical layer.

2.1.2 P2P Network Overlay Architecture

A P2P overlay network is the underlying mechanism in P2P networks [124] that is responsible for routing services and content management. P2P overlay networks can be divided into two categories of architecture: unstructured and structured.

Unstructured Overlay Networks

In the pure P2P architecture, peers and their contents are totally decentralised and loosely controlled without prior knowledge of the network structure. Queries are

forwarded through flooding mechanisms. A query session is considered complete when the list of search results satisfies the query or when the Time-To-Live (TTL) limit is reached.^{2,3} Sometimes, a complete query may be unsatisfied when the search results do not satisfy the query within TTL. Popular contents are highly replicated in order to distribute the high download request. Thus, they are easier to locate compared to unpopular contents. However, the scalability problem occurs when the size of the network increases rapidly particularly when dealing with unpopular contents where these contents are usually sparsely replicated.

There are fully decentralised and hybrid unstructured P2P overlays. An early version of Gnutella [42] is an example of fully unstructured P2P network. In hybrid or multi-tiered P2P overlays such as FastTrack [114], BitTorrent [152] and Gnutella V0.6 [155], there are some nodes which have greater responsibility than others and these nodes are usually called supernodes (sometimes called ultrapeers or superpeers). The supernodes are selected from ordinary nodes based on the selection criteria including processing capability, high bandwidth, wide access to networks and trustworthiness.

Structured Overlay Networks

Due to the scalability limitation of the unstructured networks, structured P2P networks were proposed. Structured P2P networks are designed to be scalable, fault tolerant, and self-organising. In structured P2P overlay networks, a Distributed Hash Table (DHT) is used to link up the peers and to organise their contents so that they are easier to locate. This results in an efficient routing with only $\mathcal{O}(\log N)$ hops required for lookup, where N is the number of peers. In a DHT system, each peer is assigned with a unique node identifier, and each data is also assigned with a unique key of the same identifier space. Different range of key identifiers are assigned to different peers and data with a key within a specific key range are stored at the responsible peer for the key range. During lookup, messages are propagated

^{2,3}TTL is set to avoid infinite search and to ensure that the queries are answered within a certain duration.

across the overlay to destination peers in an iterative and recursive mode—guided by the information in peers’ routing tables. Given a content with key 23, we can search the key across a network by locating the responsible peer for the key.

The growing size of the network does not affect routing performance since there is a proper management of contents and routing system. The differences among structured P2P networks are their key identifier space structures and their routing mechanisms. A few examples of structured P2P overlays are Chord [179], Pastry [160], Tapestry [218], Kademlia [128] and Content Addressable Network (CAN) [154].

2.1.3 Applications of P2P Networks

The range of P2P applications competing with client/server systems are expected to grow wider in line with the growing number of devices connected to the internet. For example, in recent years the use of P2P networks for content distribution and multimedia streaming has grown significantly. P2P has been used for commercial purposes such as for legal distribution of large files (e.g., software distribution, music and videos). Moreover, media publishers who traditionally dislike the use of P2P networks, are now turning to use P2P to get a larger audience worldwide. The applications are as follows.

Instant Messaging System

Instant Messaging (IM) is a text chatting application which allows two or more people to communicate directly with each other in real-time using P2P communication. Most of the IM applications now also include real-time voice and video chatting service and they also enables teleconferencing and direct file transfer between users. Examples of widely used IM systems are Google Chat [71], Yahoo Messenger [213], and Skype [177].

Most of the available IM systems, however, face several issues such as security risks like spam and virus attacks. For instance, Skype and Yahoo Messenger have issues with “contact request spammers”. These spammers send “add to contact

request” messages to users from bogus profiles. Once a user accepts the request, the spammer will continuously send spam messages to him/her. There were also reports on voice spam and text message spam on Skype [102, 70]. Generally, spammers may send viruses, spyware or Trojan horses through an infected file transfer or may trick users to click on Uniform Resource Locator (URL)s which then download malicious code automatically.

Distributed Storage System

P2P systems are also useful for a cheap distributed data storage. Such systems are useful for data backups through replication and erasure coding. Examples are Peer-to-Peer Archival Storage facility (PAST) (Microsoft) [61], Malugo [35], and Wuala [210]. These P2P-based data storage systems usually provide consistent data update, version control, security, and load balancing. These data are replicated at multiple different clusters to ensure the files availability at several different geographical locations. In the Malugo system, peers are clustered by geographical locality to ensure data locality. Some distributed P2P storage systems such as Malugo [35], CrashPlan [48], BuddyBackup [28] and Tudzu [191] provide a private storage option where files can be shared among known groups or organisations.

P2P Media Streaming

Media streaming is a mechanism to deliver media files which can instantly be presented to users while continuously delivered by a provider. In media streaming, video or audio are broadcast over the internet in real time. An example of increasingly popular media streaming is P2P-based Internet television (which is usually known as P2PTV) which has been widely used as a mechanism to broadcast TV channels from all over the world. Each node joining the network participates in relaying the video stream to other users. While downloading a multimedia stream, a peer is simultaneously uploading the file to another user. This provides an efficient and cheaper distribution, thus, allowing individuals to have their own TV channel.

The scalability is also improved where a large number of audiences can be reached all over the world.

The Quality of Service (QoS) of a video stream depends on the number of peers that are currently downloading the video, hence, more users contribute to a better quality video. However, broadcasters have less control in P2P-based media streaming; this hinders them from providing security and implement distribution control. It is also difficult for broadcasters to limit the access of a particular TV program regionally, to acquire data, statistics for audience research, such as show popularity and viewers' statistics according to demographics. Examples of the commercial P2PTV applications are TVUPlayer [192] and Sopcast [178].

File Sharing System

P2P file sharing applications such as Kazaa [96], Limewire [118], and BitTorrent [18] have become very popular media for users to distribute digital data such as software, multimedia, documents and digital books. Once a P2P application is downloaded and executed on a machine, the machine is connected to a P2P network. A small portion of their storage is enabled for sharing with other machines that are connected to the network. This shared folder is then filled in with files that can be shared directly among users over the network. There are several popular P2P-based file sharing systems; each uses different protocols and uses different P2P overlay networks.

Napster [33] is one of the earliest P2P-based file sharing system. The architecture of Napster is similar to the client/server network where shared files are located at multiple different peers and the server stores the index of files with their locations—in the form of IP addresses. The server refers a querying peer to the peers which have a requested file; the file is then retrieved directly from the peers using P2P communication.

BitTorrent [18] is one of the widely used P2P file sharing protocol especially for distributing large files. It was designed to provide a scalable and efficient large

file distribution without much burden to the peers and connection. Here, a swarm of peers collaboratively download and upload files from each other simultaneously. A distributed file is divided into segments, called pieces, which have an equal size for every single download. As a result, it is feasible to operate even within the network with low bandwidth and light weight processing capability. A small torrent descriptor (a static ‘meta-info’ file) is placed on a BitTorrent node acting as a tracker. A tracker has a map to the whole file and simultaneously initiates the pieces upload to several downloaders called leechers. The leechers then upload the pieces at the same time for other peers.

Another example of P2P-based file sharing applications is eMule [63] which is based on the eDonkey protocol [83]. eDonkey is a hybrid P2P network since it has a two tiered hierarchical topology: superpeers and leaf peers. Each superpeer is connected to a large number of leaf peers and other superpeers. It maintains a centralised index for the leaf nodes to which it is connected. When querying, a leaf peer sends a query to its superpeer and the superpeer returns the search results. In the case where the requested item is not available, the query is forwarded to another superpeer and these superpeers only respond if they have the search results. The superpeer responds with the list of sources for each of the chosen file. Then, the user chooses the file she/he wants to download and requests the sources for the chosen files. The client then adds the sources to its source lists and starts the connection attempt. In requesting a file, a peer initiates a direct Transmission Control Protocol (TCP) communication to the other peers in order to download a file. The latest version of eMule makes use of Kademlia [128] network in their implementation. Using Kademlia, no supernode is required and users do not have to download and to maintain the supernodes list (like the eDonkey implementation). Kademlia is fully distributed and thus more fault tolerant than eDonkey.

Recently, it was reported that the number of BitTorrent’s users has grown to over 150 million active users per month worldwide—showing the significant use of this

technology all over the world. However, in turn, they are facing challenges in managing data in the network as there is no authority to monitor the network. Unfortunately, these P2P-based file sharing systems suffer from pollution problems [111], unresolved piracy issues, and out-of-control dissemination of sensitive data [30]. This led to the ban of P2P-file sharing applications in several organisations and institutions.

2.2 Current Recognition Practices in P2P Networks

The unavailability of a single processing site imposes difficulties in implementing recognition or classification mechanism. Moreover, the distributed nature of data makes it harder to combine the knowledge from many different peers. Classification and recognition in P2P networks have been attained in many ways and predominantly using reputation-based mechanisms. In this section, we discuss the available methods for classification and recognition within P2P networks that have been found from the literature.

2.2.1 Screening

Liang et al. [113], Jia [90] proposed screening methods in determining the spam files based on their observation of the spam files behaviour (e.g., non-decodable and file's length). Liang et al. [113] classified a polluted music file by screening the files based on two rules: a file is polluted if it is non-decodable into Pulse Code Modulated (PCM) format and a file is polluted if its length is more than 10% longer or shorter than the original compact disk (CD) length. This approach only works if the official CD length is known. The spam detection proposed by Jia [90] was based on the survey work in Nguyen et al. [142]. Three features were selected to build rules in distinguishing a spam file; vocabulary size, variance of replica descriptors and per-host replication degree of a file which is identified correlated with spam. These

features were determined based on four types of spam files that were identified in Nguyen et al. [142]. The decision on rules building is made at a centralised site where all peers have contact with the centralised site for updating their rules.

2.2.2 Signature-based Approach

Signature-based approaches can be considered as the easiest and most traditional way to detect malicious peers and spam objects (e.g., music files, emails, documents, etc.) in a P2P system. In signature-based approaches, the signatures^{2.4} of spam objects or malicious peers are stored in a repository based on the users' reports. For polluted files detections, a user consults the repository before downloading files by using file signatures acquired from the search results. Examples of polluted files repositories are *donkeyfake* [58] and *Sig2dat* [176]^{2.5}. The Audible Magic [10] uses this approach to detect music piracy by storing the copyrighted music descriptions. The same mechanism is used for contact request spam in Skype, where the Skype administrator blocks or blacklists the bogus profiles based on the reports from users. Most of the signature-based approaches require the availability of a server to process the reports and to blacklist the malicious entities. Therefore, they are subject to a single point of failure risk. Moreover, they cannot only classify the “never-before-seen” objects. Hence, they are inefficient as a new version of spam object or bogus profile can be easily created and added into the P2P applications [113, 90].

However, the scheme in [220] uses a different approach where it is possible to identify if a “never-before-seen” email, is a spam by identifying the similar spam email (which has the highest number of similar features with the queried email) within a decentralised repository. Every peer (that is a part of the decentralised repository), stores features' signatures and every signature for a feature f_s is linked to a storage object which stores the Global Unique Identifiers (GUID) of emails that includes f_s . For every new received email, an user's agent (email client) queries

^{2.4}A signature is an identifier of an object or a peer—such as the hash of an object or a peer's IP address.

^{2.5}*donkeyfake* and *Sig2dat* are the polluted files repositories for the eDonkey and Kazaa users respectively.

the signatures of every feature that describe the email; the peers which receive the queries search for these signatures within their local repositories. Then, these peers respond to the user's agent with the emails' GUIDs which associated with the queried features' signatures. This results in a high storage load since peers have to store all the GUIDs for every feature in its storage; and a high communication overhead since all emails' GUIDs that associated with every match feature are sent to the user's agent—a popular feature is often associated with a high number of GUIDs.

2.2.3 Reputation-based Approach

Reputation-based systems have been used widely to detect malicious peers [94, 44, 92, 221, 51], polluted files [198, 197, 222] and email spam [52, 39]. The Eigentrust algorithm [197] calculates the authenticity of a file by having a client to collect its judgements and evaluates them based on the credibility of a client from which the judgements come. This, on the other hand, requires a large vote database which is used for vote matching and peers correlation calculation. Hence, it is resource intensive and may not be scalable in large systems [90]. The Xrep [51] broadcasts a query for the files and peers of interest throughout the network. The reputation of files and their offerers are calculated based on the responses from others. The reputation-based approach is used for email spam filtering in MailRank [39] and in Damiani et al. [52]. The MailRank collects data about trusted email addresses from different sources, calculates reputation and classifies emails according to the address of the senders. In Damiani et al. [52], email spam are detected using the digests of emails that have been detected as spam and the reputation of the reporters (the mail servers which report the spam) which are stored at a group of peers (superpeers). A mail server queries the digest of the received email to the catalogue of digests at superpeers. The superpeers then respond with the reports on the queried digests and the mail server calculates the reputation of queried digest by aggregating the reports, weighting each report based on the reporter's reputation.

The reputation approaches, however, require the reputation of peers or reputation of objects (e.g., music files, emails, documents, etc.) to be available within the system and require the tight integration of users. Therefore, they are unable to classify a new object (that “have-not-been-seen”).

2.2.4 The Intelligent Pattern Classification Approach

Pattern recognition is a method for intelligent classification. It is defined as an initiative to enable a machine to learn to distinguish the patterns of interest and make a reasonable decision about the categories of patterns based on the observation of the environment. This includes complex patterns which are unable to be classified by a human. A number of pattern classification methods have been proposed for P2P networks such as distributed Support Vector Machine (SVM) in Forero et al. [66], Ang et al. [7, 8] and distributed decision tree in Bhaduri et al. [17]. A distributed aggregation method for distributed classification is introduced in Luo et al. [125]. However, most artificial intelligence (AI)-based approaches are computationally intensive to be performed on resource-constrained devices which are predominantly prevalent in P2P systems. Thus, they are not a popular option for classification within P2P environment.

2.3 Pattern Recognition

Pattern recognition has been widely used in many different applications such as face recognition [173, 212], identifying spam [116, 135, 86], action recognition [188], and character recognition [200]. The ability to recognise in pattern recognition is analogous to how we, as human beings, learn. This is similar to how we recognise a friend even though she has a new hair colour or a new hair style. Based on our previous observations or experiences, we have memorised her features such as her face, height, walking style, and the way she speaks; the only feature that is different is her hair. There is also a situation where we fail to recognise an old friend whom we

have not met for a long time. This happens when our memory starts to deteriorate or whenever there are significant changes to her features such as weight, hair or face. These changes do not create a new person. In pattern recognition, these changes or differences are called distortion of the original pattern. Pattern recognition should be able to recognise the patterns even from the distorted patterns based on the knowledge of the original patterns.

There is always such a need to make decisions, diagnoses and judgements in our lives using previous experiences or evidence. Dealing with complex and large information makes it very difficult to make such decisions. Some decisions may be able to be performed manually by humans. However, this may take a considerable amount of time to complete. In some cases, a problem can be too complex, poorly understood, and has a large amount of data with implicit information. Such problems may be impossible to be classified or understood accurately by human. The areas of artificial intelligence brought intelligent machines which are capable of extracting essential knowledge from the information, learn from it a representation of a concept of category and identify or classify objects.

In decision-theoretic pattern recognition approach, objects are represented as points in a feature space and the coordinate axes correspond to observations of features. There are two techniques in this approach; deterministic and probabilistic techniques. In the deterministic approach, it is assumed that each unknown instance has an unambiguous pattern class associated with it. An example of the deterministic approach is k -Nearest Neighbours (k -NN) [62, 134]. In probabilistic techniques, the optimal decision is assumed to be made by reasoning the probability distribution of the problem domain. As in real-world problems, this is useful in dealing with noisy features and overlapping classes in feature space. Examples of probabilistic techniques are Bayesian learning methods [105, 134].

The connectionist approach, is widely known as the artificial Neural Network (NN) inspired by the model of neurons in the human brain. The human brain has

highly complex, distributed, non linear and massively parallel processing. Each processing unit is called a neuron; the connection of neurons in the brain is massively parallel. This makes a human brain performs significantly faster than a computer. Even in the era of parallel computers, it is difficult to achieve this high degree of parallelism using computer software. However, NN algorithms mimic small part of this functionality by interconnecting a network of nodes. Examples of connectionist algorithms used for pattern recognition, include the Hopfield network [85] and Bidirectional Associative Memory Network (BAMN) [106].

In general, pattern recognition algorithm can be categorised based on either it uses labelled or it uses unlabelled training data. It also can be distinguished based on its operation. The pattern recognition algorithms are commonly classified using three methods as described below.

2.3.1 Eager and Lazy Learning

Firstly, pattern recognition algorithms can be classified as eager or lazy learning. This classification depends on when the model is built—either during training or during classification [134]. In lazy learning (e.g., k -NN), training examples are simply stored and generalisation is postponed until classification. Therefore, the training overhead is usually cheap, while, the classification overhead is more expensive. For each classification of an instance, the relationship of the instance with the stored examples is examined. Assuming that instances can be represented as points in Euclidean space, a decision is made locally—it only involves the stored instances that are close in distance with the instance to be classified. Most of the available methods (e.g., SVM [29], NN [132] and naive Bayes [62]) are eager learning approaches since the generalisation is performed during training. The advantages of lazy learning are that it has a fast training time and does not require retraining the network when a new pattern is presented. There is no need to generalise the target function using the whole training dataset. Instead, the generalisation is performed locally for each classification query [134].

2.3.2 Supervised and Unsupervised Learning

Secondly, we can distinguish these algorithms based on whether they are supervised or unsupervised learning. Supervised classification is a process to determine if an object x is in category B , given previous examples of category B . A supervised classifier is designed using class information (labelled training data) to discriminate objects. All available supervised pattern algorithms involve two phases; learning and recall. The learning phase involves training the system with examples of patterns that belong to a certain class for recognition. The recall phase, on the other hand, involves recognising the class of a pattern. Examples of supervised learning algorithms are k -NN, naive Bayes, and SVM. Unsupervised classification is a process to group objects with similarities without any class information (unlabelled training data). The classification evolves on its own. Clustering—(e.g., Self Organising Map (SOM) [103] and k -means [81])—is an example of unsupervised classification.

2.3.3 Batch and Online Learning

Pattern recognition algorithms may also be classified as batch or online learning—this depends on the samples involved during each learning process [164]. Batch learning is performed using the whole training set simultaneously. In online learning, learning is performed after presentation of each training example. Batch learning is inefficient in training large training data and in systems with frequent updates. It is prone to local minima [164] and requires retraining on the frequently changing training dataset. However, it is faster for a system with small training sets.

The focus of this thesis is to develop a supervised learning algorithm over a distributed P2P network. Lazy and online learning are suitable for data within P2P networks which are rapidly changing [164].

2.4 Distributed Pattern Recognition

Given the distributed nature of information today, additional research on distributed pattern recognition is needed rather than focussing on the centralised pattern recognition. Several works have been performed on distributed pattern recognition algorithms for many different distributed computing platforms such as neurocomputers (also known as parallel computers) [183], grid computing [138], multi-core [41], wireless sensor networks (WSN) [100, 140, 101], and P2P networks [125, 17, 7, 66] from the literature. Let G_P be a P2P network of N peers. A distributed pattern recognition system within a P2P network may be well-structured into a fixed architecture and may involve only a subset of peers of G_P . It may also be an unstructured system with dynamic architecture that expands and shrinks depending on the network and it involves all peers in G_P . The former is classified as a cluster-based distributed pattern recognition algorithm, while, the latter is classified as a network-wide distributed pattern recognition algorithm. In this section, we briefly discuss the components of distributed pattern recognition and this is followed by the discussion of several available distributed pattern recognition algorithms.

2.4.1 Distributed Pattern Recognition Components

Distributed pattern recognition systems are basically involved four main components: 1) data collection and feature extraction, 2) learning, 3) prediction and 4) aggregation. The distributed approach can be applied to any or all of these components. For example, a distributed pattern recognition which only requires a centralised data collection and feature extraction, may perform a fully-distributed learning and classification. The data collection step involves raw data collection from centralised or distributed observation (e.g., Internet, sensors, cameras, etc.) and the feature extraction step includes data analysis and training dataset generation. The number of samples to be collected and the sampling period (e.g., limited period or continuous) may vary according to the classification problem. For a distributed observation, the collected raw data can be sent to a central site for feature

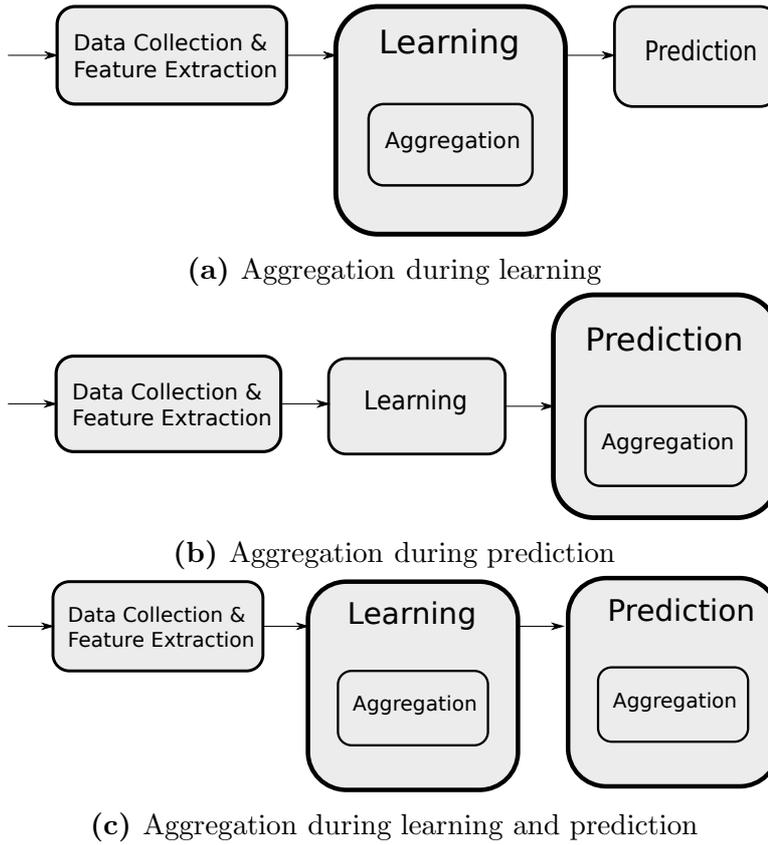


Figure 2.2: Aggregation can be a part of learning as shown in Figure 2.2a; a part of prediction as shown in Figure 2.2b; or a part of learning and a part of prediction as shown in Figure 2.2c.

extraction. However, the more efficient way for a large P2P network is to perform the feature extraction locally at each peer, considering the volume of the raw data to be transmitted can be very large. The generated datasets have a predefined format (depending on classifiers' requirement) and consist of a number of attributes. Rich and complex data is usually represented using a high-dimensional dataset format.

The learning component involves presenting the generated dataset into a learning algorithm which then performs generalisation by building a classifier. Some examples of learning algorithms is Backpropagation Neural Network (BPNN), naive Bayes, k -NN and BAMN. In distributed pattern recognition, multiple different classifiers are built at multiple different sites. Thus, an aggregation component involves combining these local classifiers to build a single classifier which represents a global model, or combining predictions from these local classifiers to build a single prediction which represents a global prediction.

The aggregation can be performed during learning to build a global classifier or it can be postponed during learning but is performed to reach a global result during prediction (see Figure 2.2). In case an aggregation for a global classifier has been built during the learning phase, there is no communication during prediction. In Luo et al. [125], there is no aggregation during learning to avoid an expensive communication cost during global classifier building. Instead, the aggregation is performed during the prediction phase when the local prediction results from all sites are combined into a global prediction result. In some algorithms [140, 8], aggregation is applied both during learning and prediction.

In contrast to aggregation during learning, which requires a higher communication cost during learning than during classification phase, aggregation during prediction requires a higher communication cost during classification than during learning. Most of the available pattern recognition algorithms require several cycles of learning before converging to the global optimum. Thus, when an aggregation is performed during learning within a large network, it results in a significant communication cost. The aggregation during learning in these algorithms is therefore inefficient for a domain with frequent data update compared to aggregation during prediction.

2.4.2 Artificial Neural Networks

An artificial neural network (NN) itself is a readily distributed, decomposable algorithm. Therefore, any NN model can be directly distributed by mapping each neural network unit to a processor. An NN is a fault-tolerant system [185, 16, 137]—whenever any connection between nodes has failed we can still obtain the result, albeit with a graceful degradation of the system, with respect to the degree of malfunctional nodes or connections. In an NN, the output is not stored at a single location (neuron) but it is a combined result from information at different neurons throughout the network [185, 137].

Perceptron is an NN introduced by Rosenblatt [158]. The early perceptron models only had an input and output layer. Each neuron receives inputs from input

neurons and produces output signals from these input signals. The single perceptron is the simplest form of NN needed for classification of linearly separable patterns. Given η be a positive constant, called a *learning rate*, and $0 < \eta < 1$. The η value controls the degree of weight increment or decrement for each learning iteration. The perceptron training rule is proven to converge to local optima in a linearly separable decision space, provided that the value of η is sufficiently small [132]. Hence, it has a limitation since it is only able to converge whenever the training data are linearly separable [132].

Therefore, a multi-layer perceptron network, that is the BPNN model was introduced by Rumelhart et al. [163] to solve the limitation by using hidden layers and non linear activation functions. The BPNN algorithm is a supervised, feedforward multi-layer perceptron architecture that is applicable for non linear decision spaces. Typically, a BPNN has d inputs and c outputs where d is the number of attributes and c is the number of classes. BPNN aims to find the best weight values which can give a best-approximate minimum error from the decision space. The base of this algorithm are delta rules and gradient descent. The delta rule aims to get the best-fit approximation to the target concept [134] through finding the best weight from the hypotheses space of potential weights.

Another example of NN models is the Radial Basis Function Network (RBFN) [150] which provides a global approximation to the target function through linear combination of several radial basis functions. The basic form of a RBFN is a three layer structure; the first layer is an input layer, the second layer is a hidden layer which involves a non-linear transformation from input space to hidden space, using Radial Basis Functions (RBF), and the third layer is an output layer which provides the response. A RBF usually uses delta rules in updating the weights towards minimising error. In contrast to BPNN, in RBF models, the weight for connections from input to hidden layer are not changing. RBF networks are more efficient than BPNN in dealing with high multi-dimensional problems as the input layer and the hidden layer are trained separately [134]. However, it is highly sensitive to the number

of centroids and their location within decision space. Improper selection of these values may cause inaccurate or inefficient classifications as a RBF network is highly sensitive to these parameters particularly to the location of centroids. Choosing the k -value is a difficult problem for well separated clusters and even more difficult for multi-dimensional problems [77]. An efficient method to select k is presented in Hamerly and Elkan [77]; it uses principal component calculation which requires $\mathcal{O}(nd^2 + d^3)$ time and $\mathcal{O}(d^2)$ storage where n is the total number of training instances and d is the number of attributes.

Associative Memory (AM) is a class of NNs inspired by a biological brain system by recalling memory of a particular item, relating the memories from the previous experiences. The data is retrieved based on its content rather than through addresses (e.g., computer memory). Like synapses in the brain, all memories are distributed throughout the network, and these are associated with each other. By combining these pieces of information, a memory is obtained. Thus, AM is very useful when we are looking for an optimal match given partial information.

The memorisation and recall of these patterns are interrelated. AM provides a mapping of an input vector to its corresponding response pattern. The decisions on the outputs are dependent on the interconnection between input information and output. For an input \mathbf{x}_k for k th pattern, the expected output is \mathbf{y}_k . Thus the AM models aim to learn this association. One of the advantages of AM is a high degree of resistance to noise as the neurons could be operating under noisy conditions. Some examples of AM algorithms are the Hopfield network, BAMN and Morphological Associative Memory (MAM).

The Hopfield network [85] is an NN model with a single-layer recurrent network architecture. In recurrent architecture, the memory is stored in a series of events where the memory of the network unit at event k is used as an input to another unit at event $k + 1$. The neurons in a Hopfield network are binary threshold units where a unit can only have two states *ON* (+1 or 1) or *OFF* (-1 or 0). The local minima in the energy function corresponds to the memory of trained patterns. A

Hopfield network is guaranteed to converge to a stable state and, therefore, there is a limit to its memory. A network with n_{total} neurons can store only approximately $0.15n_{total}$ training examples [85].

Another widely known AM algorithm is the Bidirectional Associative Memory Network (BAMN) which has a four layers architecture. The first layer is the input layer, the second and third layers represent Bidirectional Associative Memory (BAM), and the final layer is the output layer. The BAMN architecture is shown in Figure 2.3. Every neuron in the first BAM layer (second layer of BAMN) is

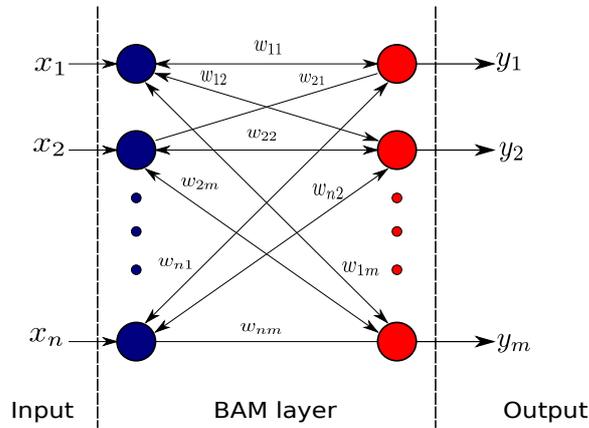


Figure 2.3: An example of BAMN architecture with n input neurons and m output neurons. w_{ij} is the strength of connection from i th input neuron to j th output neuron.

connected bidirectionally to the neurons in the second BAM layer (third layer of BAMN). The first and second BAM layers may have a different number of neurons. BAM layers represent the memory matrix of this model and the bipolar Hopfield rule is used for learning the weights.

NNs can adapt their free parameters to the changes in the surrounding environment. This is usually done by adjusting the free parameters^{2.6} in the learning mechanism. However, this adjusting procedure has to perform several iterations to obtain suitable parameters that can lead to the output closest to the desired output.

^{2.6}The strength of the connection. In human brain, this refers to synaptic connection.

In gradient descent, a large learning rate η may result in a fast consensus reached with poor classification performance, where too large η may lead to an “under-damped” case, which means it will have an “overshoot” or “undershoot” of the point of minima. In other words if η is greater than a certain value, the learning can be oscillatory. If η is too small, it will lead to “over-damping” where the convergence will be reached at a large number of iterations. The large η during training in a gradient descent algorithm reduces the convergence rate, while low learning rate leads to the frequent weight update (iterations). The frequent weight update increases the communications of neurons between layers. In NN models, complex problems may involve a massive number of processing units or neurons [185]. There is good degree of parallelism involved and most units can perform processing independently. The connections among a large number of neurons are highly used, therefore it requires a high interconnectivity between nodes.

2.4.3 Ensemble Classifiers

Most of the available semi-distributed PR algorithms are ensemble classifier approaches. The idea of an ensemble scheme is to combine multiple learners and combine their predictions to achieve a high classification accuracy. This involves manipulating the training samples using several sampling approaches. The base classifiers are built from local data and the outputs from these base classifiers are gathered at a central node as the meta-level data. The central node then generates the meta classifier using the input from this meta-level data. The most common meta-classifiers are arbiters, combiners [34], boosting [169], bagging [23] and stacking [209].

The meta-level data for different ensemble methods are usually produced in different ways. In arbiters, it is built by selecting the difficult-to-classify training examples from base classifiers. These training examples are from the predictions with no

majority decision from any base classifiers (i.e., the predictions with confusing decisions), and these conflicting predictions are resolved by an arbiter. The meta-level training set in combiner approach is produced by the *composition rule* [34].

Boosting [169] constructs a classifier with higher accuracy from weak classifiers. The learning in boosting starts as a weak classifier on distributed training data and better classifier models are built iteratively. Each model is trained successively on a dataset with out-of-bag error (misclassified instances) by the previous model, and these models are given weight according to their success in predictions. The successive learning is executed using the same training set. Finally, their outputs are aggregated using voting and averaging. Essentially, boosting focuses on difficult-to-classify training examples by giving more weight to these examples so that they have higher probability to be chosen during sampling. AdaBoost [67] and Adaptive Resampling and Combining (arcing) are two of the well known boosting algorithms. The composite classifiers in boosting basically reduce the variances significantly. The error is reduced by using the combination of many hypotheses with large errors [157].

The large number of classifiers can reduce the time complexity of an individual classifier ^{2.7} but linearly increases the overall processing time in boosting. Since the base classifiers are trained in a serial fashion, the convergence is slow and inefficient for P2P environment. The iterative nature of boosting may also cause a high communication load. Other drawbacks of boosting include a large number of iterations and it suffers from the over-fitting problem [153].

Bagging [23] uses multiple bootstrap sampling-by-replacement of a training dataset to create several different datasets. Then, it learns the datasets on the separate learning models. In contrast to boosting, classification for each classifier in bagging can be performed simultaneously, as the training samples for each classifier are selected independently. The outputs of these models are then combined by uniform voting or averaging (for regression) to form the final output. In bagging, bias and variance of unstable classifiers (e.g., NN and decision trees) can be reduced using

^{2.7}The training dataset per classifier is smaller when a larger number of classifiers is used and vice versa.

voting or averaging from many local classifiers. However, the performance of some stable classifiers^{2,8} (e.g., k -NN) are slightly degraded due to the use of small training sets [23, 14, 153].

Stacking is a meta learning scheme that works by combining different types of model. In stacking [209], a training dataset is partitioned similar to the cross validation approach. It is first split into two disjoint sets. The first set is used for training on several base learning models and the second set is used for testing. Then, the prediction result is used for inputs to train the higher level learner and the correct recalls are used as the outputs. In Ho [84], each learning model is trained on randomly chosen subspaces of the input feature spaces and all the participating learners are then combined by using voting.

2.4.4 Graph Neuron-based Algorithms

The Graph Neuron (GN) [99] is a graph matching based pattern recognition designed for wireless sensor environments. The approach performs single-cycle learning, therefore, the classification computation is efficient in dealing with frequent data updates. It is also lightweight, accurate and scalable for a network of resource-constrained devices. Its structure is a connected graph with a set of vertices, V and a set of edges, E . The network represents all possible points in representing a pattern. Each node, called a *neuron*, holds the $\{u, s\}$ pair information, where u represents the domain value associated with the neuron and the s represents the position of the neuron in the pattern space.

Figure 2.4 shows the GN structure as introduced in Khan [99] where we consider an input pattern of size 7 bits, with input values $\{0, 1\}$. The dotted line represents the inter-node communication of the pattern **0100111**. Each neuron communicates with its adjacent nodes (i.e., its left and right neighbours). The value corresponding to a specific position in the input pattern is mapped and compared with the neuron at the same position (which holds the same value) for memorisation (signifies a new

^{2,8}A stable classifier is a classifier with invariant error to the small changes on training samples

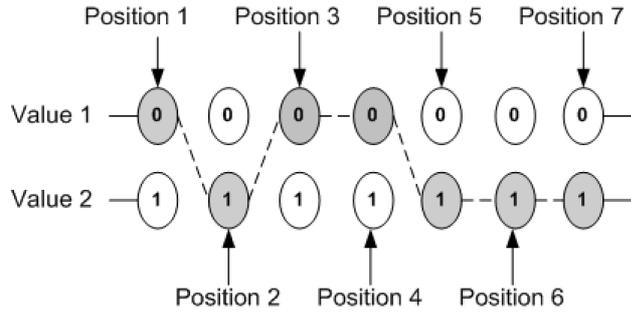


Figure 2.4: A GN structure for a binary pattern with domain values: $\{0, 1\}$ and pattern size 7. The possible values was represented by the number of rows while the size of the pattern is represented by the number of columns.

input pattern) or recall (signifies an old pattern). This scheme has been applied in WSN applications where its quick recall time and high scalability characteristics have been proven [101].

The Hierarchical Graph Neuron (HGN) [140] is a variant of GN algorithms that was designed to solve the accuracy limitation in GN. In this approach, a group of nodes is arranged in a fixed hierarchical topology as shown in Figure 2.5. These nodes are reactive and process the incoming input in parallel. The input of the nodes from the upper layer is the combination of output from the nodes (at the same column) at the lower layer and the outputs from its left and right neighbours. The HGN, however, requires a base station to interpret the final outputs from the top nodes. The HGN [140] is scalable to multi-dimensional problem at high expense—a large number of nodes is required. The number of nodes in HGN is $u \left(\frac{d+1}{2} \right)^2$ where u is the size of domain (e.g., 2 for binary data $\{0, 1\}$) and d is the number of attributes (data dimension).

2.4.5 Network-wide Algorithms

The training data in P2P classification algorithm is distributed across the network and the global classifiers are built at every site. There is no entity to coordinate the recognition process. A network-wide algorithm requires a frequent information exchange among nodes to ensure the local models accurately approximate the global

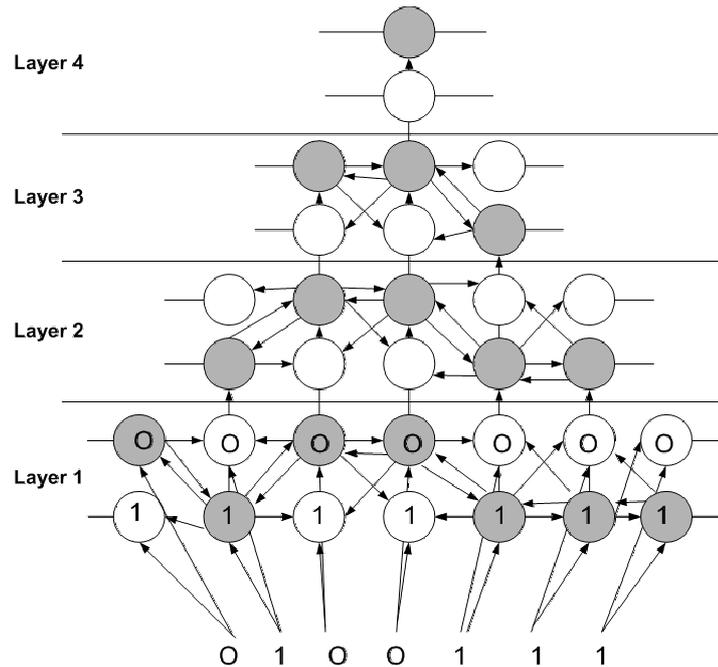


Figure 2.5: An example of HGN structure for binary pattern: “0100111”. It has four layers where each layer (except layer 1) has less two columns than the lower layer and nodes at the top have only one column.

model. This exchanged information can be local models, statistics or training instances. Therefore, another important concept in a distributed pattern recognition is the data integration aspect that specifies information to be exchanged among nodes, the destinations where each node has to send information to, and how the available information can be combined to produce the final classification decision. In this thesis, we distinguish the network-wide algorithms into two categories: complete graph classification algorithms and local classification algorithms.

Complete Graph Classification Algorithms

In a complete graph classification algorithm, each node broadcasts its model, statistics, or training instances over the network. Thus every node is connected to all nodes and builds its own combined classifier. Two examples of complete graph algorithms are Distributed Boosting [108] and P2P Cascade RSVM [7].

Ang et al. [7] introduces the P2P Cascade RSVM which applies Reduced Support Vector Machine (RSVM) algorithm [110] on each peer’s local data to produce

a local model; the model is then broadcast to all other peers. Every peer combines the received models with its local model to produce a better local classifier. The RSVM is used instead of traditional SVM as the base classifier in the P2P Cascade RSVM [7]. The RSVM is an improved version of traditional SVM with smaller number of support vectors. Hence, it requires less local computation and communication overhead in exchanging models among peers than that of the SVM. Nonetheless, the communication overhead of the P2P Cascade RSVM is still expensive ($\mathcal{O}(N)$ where N is the network size) since these models are exchanged among all sites. Moreover, frequent model exchanges are required since these models need to be reconstructed when dealing with a frequently updated training set. The RSVM approach has been empirically evaluated and it is proven to provide an accurate model even with only a small subset of the training set [110].

Although the RVSM is less expensive than the traditional SVM (since it uses smaller number of support vectors), it still requires some basic, expensive SVM computation. Generally, verifying the optimal solution to SVM problems requires $\mathcal{O}(n^2)$ dot products, while solving the quadratic programming problems requires $\mathcal{O}(n^3)$. The memory requirement to store the dot products is $\mathcal{O}(n^2)$. The test time complexity is $\mathcal{O}(s_v)$ where s_v is the number of support vectors. The s_v in the traditional SVM is at least $n \times \epsilon_n$ where ϵ_n is the training set error rate. This s_v , however, is reduced in the RSVM. In SVM, different types of kernel or the same kernel with different parameters embed data in different high-dimensional space. The linear decision boundary which SVM finds, thus, depends on the choice of kernel and its parameters. Hence, the performance of SVM depends on these decisions [170].

In Lazarevic and Obradovic [108], a distributed boosting algorithm is proposed where all local datasets from all different sites are merged to preserve a similar sampling outcome as in the centralised boosting. This approach, however, consumes a high communication overhead as the local sum of weight vectors and learner model are broadcast across the network for each iteration. Hence, it requires $\mathcal{O}(N)$ communication cost for each boosting iteration.

Local Classification Algorithms

In a local classification algorithm, as shown in Figure 2.6, each node calculates the recognition results locally within its neighbourhood. The original concept of local algorithm arose from the distributed aggregation problem—each peer aims to achieve the approximate global information through a magnitude of communications that is bounded by the number of neighbours it has [54]. Local algorithms [54] are well-known as an efficient distributed aggregation algorithm. It requires each node to communicate only with its immediate neighbours. The training data in P2P environment is frequently changing; a P2P-based algorithm requires frequent model rebuilding using incremental algorithm to ensure the current model is up-to-date with the current data. Model rebuilding is a resource intensive task in pattern recognition. Thus it is suggested that this task is only performed whenever the changes of data results in the model no longer represents the current data. Local algorithms require the worst case of $\mathcal{O}(N)$ communications.

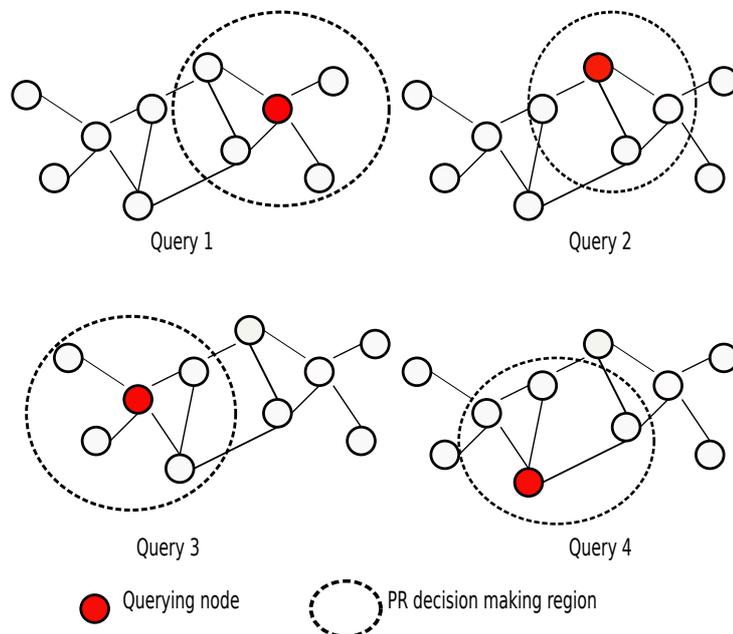


Figure 2.6: An example of local classification algorithms with four different queries submitted from four different nodes. The red node represents the node which submits a query and the dotted circle region represents a Pattern Recognition (PR) decision making region. Any node in the network can act as coordinator to process its own query based on information from its neighbourhood.

Distributed Ivote with Distributed Plurality Voting (Ivote-DPV): In Luo et al. [125], a P2P classification is presented where the local classification at each peer was implemented using Ivote approach [24]. In the Ivote-DPV, several local classifiers are built at a single site to avoid the high communication overheads due to the frequent model exchange. Each local classifier is built iteratively by selecting random small pieces of data, building several base classifiers using the data, and then pasting all the base classifiers together. Distributed Plurality Voting (DPV) [125] is proposed to aggregate results from several sites in models monitoring and during classification. To minimise the communication cost, DPV also includes a condition check protocol before exchanging predictions to limit communications among peers. For difficult problems, this approach may incur a high communication overhead and a long classification time [7].

Method of Multipliers-Non Linear Distributed Support Vector Machine:

In the Method of Multipliers-Non Linear Distributed Support Vector Machine (MoM-NLDSVM) [66], Alternating Direction Method of Multiplier (ADMM) is used to support decomposition of convex optimisation problem in which solutions from sub-problems are coordinated to seek for a global solution. ADMM is useful for consensus problems that aim to find a solution for distributed optimisation. MoM-NLDSVM is a fully decentralised system which can be classified as an epidemic-based model of a distributed system. In an epidemic-based distributed system, each node continuously exchanges information within its neighbourhood where this information becomes an input for its local function until a termination condition is satisfied. Let J be the number of nodes in the system. Given a training dataset for node j , $S_j = \{(\mathbf{x}_{ji}, y_{ji}) : i = 1, 2, \dots, n_j\}$ where $\mathbf{x}_{ji} \in \mathcal{X}$ and $\mathcal{X} \subseteq \mathcal{R}^p$. The neighbourhood of node j is Γ_j . Weight and bias for unit j is $\{\mathbf{w}_j, b_j\}$. To force these variables to achieve consensus, the consensus constraint is included. Letting $\phi : \mathcal{X} \rightarrow \mathcal{H}$ is the projection from dimensional space \mathcal{X} (rank p) into higher dimensional space \mathcal{H} (rank P).

In order to reduce the computational complexity in space \mathcal{H} , the computation is performed in a reduced rank space L where $L < P$. A set of $\mathfrak{X}_l \in \mathcal{X}$ is selected where $l = 1, 2, \dots, L$. Each node j has its own local discriminant function g_j^* on the space rank L . Each $\mathfrak{X} \in \mathcal{X}$ corresponds to $\phi(\mathfrak{X}_l) \in \mathcal{H}$ at optimal $\{\mathbf{w}_j^*, b_j^*\}_{j=1}^J$ of the convex optimisation problem—the solutions are expressed in terms of a kernel. Each node calculates its kernel locally provided that the type of kernel $\mathcal{K}(\cdot, \cdot)$ is common to all nodes. The sent message to its neighbour is $\tilde{\mathbf{w}}_j(t+1)$ and $b_j(t+1)$ which corresponding to $(L+1) \times 1$ vector. Finding the $L \times 1$ vector $\tilde{\mathbf{w}}_j(t)$ from $\mathbf{w}_j(t)$ costs $\mathcal{O}(L)$. The communication cost of a node at each iteration is $\mathcal{O}(L\Gamma_j)$.

The performance of distributed SVM in MoM-NLDSVM [66] depends on the selection of L and $\{\mathfrak{X}_l\}_{l=1}^L$. The decision on this selection depends on the available training data S_j at node j , prior knowledge of of the discriminant function and the type of kernel used. Small value of L reduces the computational complexity and communication overheads. However, this results in a high disagreement of local discriminant function between nodes [66]. Solving the optimisation solution and selecting a suitable kernel and L are not trivial. Fine tuning of these parameters may provide a model with a desirable trade-off of speed of convergence, performance, computational complexity and communication overheads.

P2P Distributed Decision Tree (PeDiT): A decision tree for a horizontally distributed data within P2P networks, that is the PeDiT, is proposed in Bhaduri et al. [17]. The Iterative Dichotomiser 3 (ID3)-like decision tree is induced from the entire dataset and all peers jointly build identical decision trees instead of several weak classifiers as in Luo et al. [125]. This approach offers low communication overheads and works with lack of synchronisation. It handles data changes and dynamic network seamlessly, thus making it suitable for large and dynamic networks. It is built based on the Distributed Majority Voting (DMV) [207] (refer to Section 2.5.7) which determines a condition when a peer needs to send messages to its neighbours. For simplicity of computation in distributed setting, misclassification error is used as an impurity measure instead of well-known methods such as information gain and

Gini information gain. The depth of the tree also limits to a predefined value [17]. Messages are only sent whenever there are any changes to one of majority voting instances that may impact on the results. This approach requires $\mathcal{O}(d^2)$ condition checks where d is the number of attributes and these can be reduced to $\mathcal{O}(d)$ through pivoting. In the PeDiT, comparing two attributes can be achieved in $\mathcal{O}(N)$ where N is the network size.

2.5 Distributed Aggregation

Given a network G_P , a peer p_i , has a value b_i and $\forall p_i \in G_P$. The aim of a distributed aggregation is to calculate global statistics. The distributed primitive operations such as *max*, *min*, *average* and *sum* are used to combine results from distributed sites. These operations basically produce a single output value which summarises the values from a set of inputs [31]. The distributed aggregation problem for a distributed pattern recognition is a problem during learning and classification is as follows:

- (a) ***During learning*** - to obtain a global classifier \mathcal{C}_{G_P} which represents the generalisation of training datasets from all nodes in G_P , $\mathcal{C}_{G_P} = \mathcal{C}(\mathcal{D}_{G_P}) = \bigcup_{i \in G_P} \mathcal{D}_i$ where \mathcal{D}_i is the training dataset at node i ; or $\mathcal{C}_{G_P} = \bigcup_{i \in G_P} \mathcal{C}_i$ where \mathcal{C}_i is a local classifier at node i .
- (b) ***During classification*** - to obtain a global classification result \hat{R}_{G_P} which represents the generalisation of local results from all nodes in G_P , $\hat{R}_{G_P} = \bigcup_{i \in G_P} \hat{r}_i$ where \hat{r}_i is a local classification result at node i .

As the classifier models or classification outputs usually have more complex outputs that have multiple values (e.g., multi-dimension and multi-class), it is therefore more complicated to compute than other single valued aggregation problems.

Distributed aggregation may be used in distributed classification during querying, model monitoring and training. As the global training dataset in large distributed system is hard to be determined, distributed aggregation within P2P networks is usually aims to obtain an approximation of global statistics instead of exact values. Distributed aggregation has been used widely to approximate a global statistics for clustering, associative rule-mining and classification within P2P networks [54, 107, 207, 125].

2.5.1 Sequential-Reduction

In a sequential-reduction, the training data or local results from every peers are combined sequentially at central site. Hence, the aggregation time complexity is $\mathcal{O}(N)$ for a network which has N peers. Most cluster-based distributed PR algorithms such as combiner [34], boosting [169], bagging [23], stacking [209] and Map-Reduce based machine learning [41] use sequential-reduction in combining their local results or classifiers.

2.5.2 Parallel-Multilayer Reduction

An example of a distributed pattern recognition which performs parallel-multilayer reduction is the HGN [140]. In the HGN, the local outputs from each level are combined in a hierarchical structure where given a root node is at layer 0, each node at position pos_j at an upper layer l_i combines the outputs from three nodes. These three nodes are a node at position pos_j at its lower layer l_{i+1} ; and its adjacent nodes (node at pos_{j+1} and node at pos_{j-1}) within the same layer. The results are combined iteratively until they reach the root node at layer 0. Therefore, each node only communicates with other three nodes which, given communication complexity per peer $\mathcal{O}(3)$ and the aggregation time complexity, is equal to $\mathcal{O}(NumLayer)$ where $NumLayer$ is the number of HGN layers. In the HGN implementation, however, an active node at each position also sends its local result to a base station and thus communication cost is $\mathcal{O}(|G_R|)$ where G_R is a set of nodes in the HGN.

2.5.3 Aggregate-Broadcast

Distributed aggregation during the learning phase may involve all nodes transmitting local statistics or data to the centralised node, which then builds a global model and broadcasts the model over the network. Data in P2P networks are rapidly updated at multiple locations and this demands incremental learning instead of by-batch learning. Therefore, the high communication overhead in transferring these models is impractical. Distributed aggregation during the classification phase can be performed by lazy learning which does not require any model building. The easiest way to perform aggregation in this condition is to send all local results to the centralised node where aggregation is performed—provided that a ‘server-like’ entity exists in the system.

An example application of the aggregate-broadcast method is Babcock and Olston [11] which monitors the answers to continuous queries over data streams from distributed sites. The answers are calculated by a central node which then distributes them to all sites. The nodes from all sites then validate the answers. In case an answer is found to be false by any node, then an alarm is then sent to the central node. The central node then updates the answer and re-distributes the new answer to all sites. This process is performed repeatedly until there is no alarm received from any node (converge to correct answer). This approach, however, does not scale well and is prone to a single point of failure. Hence, current research attention focusses on investigating an efficient and effective fully-distributed approach to combine local results from multiple sites.

2.5.4 Flooding

Flooding is one of the most popular mechanism for query processing within unstructured P2P networks. In the flooding-based query processing, a node floods the entire network with the query so that every other node within the network can process and give the query results. TTL is given to limit the flooding area (see Section 2.1.2). In the flooding-based approach for a classification problem, a node may flood the

entire network each time it has new training data so that other nodes in the network have the replica of the data. Thus, for each node, the communication overhead per update grows linearly with the size of the network. This approach is definitely inefficient for P2P networks having millions of nodes. Therefore, randomised-flooding can be used where a node floods the data to several nodes using a random walk.

2.5.5 Gossiping

In the gossip-based approach, the computation is performed locally and recursively from a node to another node from time to time. This involves gossiping where every node passes its statistics or training instances to a random node in multiple rounds. This epidemic-based model is proven to converge to an exact value if it is uninterrupted. Examples of epidemic-based model are gossip-based randomised model [97, 22], *newcast model of computation* [107] and averaging model [88].

In the gossip-based model, each peer chooses other peers at random to ensure peer diversity and applies the same protocol with others to average its value with other peers' values. This is performed periodically and with limited communication and processing at each round. However, it may require expensive communication whereas involving hundred of messages per node to calculate this statistic [133]. The problem of gossip-based models is that they are slow—they need time for convergence [53] and require resources that scale directly with the size of the system [54].

2.5.6 Local Algorithm with Random Walk Sampling

In Das et al. [53], a local algorithm using a random walk operates in identifying the top l inner products among pair of feature vectors in P2P networks. This approach works by random sampling the horizontally-partitioned distributed data without traversing all the nodes in the network for collecting data. Let n be sample size and m peers to visit for every inner product entry i where i is an element of top l inner products. $\mathcal{O}(mnl)$ independent random walks are launched where the maximum number of hops is $\mathcal{O}(\log N)$ peers for each random walk [53]. In contrast

to the randomised-flooding approach (where a new instance is sent to other random nodes), in random walk sampling, the instances are collected from the random nodes.

2.5.7 Distributed Majority Voting

Distributed Majority Voting (DMV) [207] is an associative rule mining technique which was proposed for aggregating single values within a large network. It is a reactive algorithm where messages are only sent based on the condition checked result. It is described as follows. Given a peer, P_i , has a real number b_i , and a threshold $\theta > 0$. Let the same threshold apply at all peers. The aim is for the peers to collectively determine whether $\sum_i b_i$ is above $N\theta$ where N is the number of peers in the network.

In the Local L2-Thresholding (LL2T) algorithm [208], the DMV is used to monitor the changes of local data to ensure that the current model represents the global estimates of current data. If the local statistics no longer represents the model, the monitoring service then will alarm the system to update the global statistics. The update, however, is executed at a server which then broadcasts the updated statistics to all nodes. The idea is that the model will only be rebuilt whenever it no longer represents the current data. This is to avoid the unnecessary use of resources for rebuilding the model if it can. However, the monitoring cost can sometimes, be high.

The DMV is a local algorithm which has been used widely for aggregation in P2P data mining [53, 17, 208]. It works well in a simple problem but for a problem with many entries (many different values to aggregate), a separate DMV problem has to be calculated for each case or value. As a result, the communication cost increases with an increased number of cases or values [53, 17].

2.5.8 Distributed Plurality Voting

Distributed Plurality Voting (DPV) is proposed in Luo et al. [125] to obtain global classification results for a one shot query or distributed classifier monitoring within

P2P networks. This approach is similar to DMV which is a reactive algorithm with small modifications as follows. DMV is a single-valued function which uses a threshold to determine whether the value is agreed upon by the majority of peers, while DPV is a multi-valued function to calculate the option with majority votes. However, DMV is equal to DPV for the two options problem [125].

2.5.9 Best-effort Algorithm

Best-effort algorithms are widely used in WSN query processing [43, 74, 126, 195]. In this approach, a node broadcasts a query to build several spanning trees. Then, results from different nodes within the trees are combined from bottom to the top of the trees. Any failed node may cause an incorrect result, and to solve this, a validation metric is used. For example, Gupta et al. [74] used the *Completeness* validity measure, whereas, Considine et al. [43] used *Relative Error*. Completeness refers to the percentage of nodes whose data contribute to the query result. Relative Error is calculated by $|\frac{\hat{x}}{x} - 1|$ where \hat{x} is the reported result and x is the true result. These calculations can only be determined by an Oracle which has a perfect knowledge of the network. Bawa et al. [15] proposed the *single-site validity* metric to validate the result. It is based on a view from a single node, where the sum and count queries are subject to the network behaviour model. However, this metric can be resource-intensive to apply when the network and data are dynamically changed. It was reported in Bawa et al. [15] that it may cost five times higher than the most efficient best-effort algorithm.

2.6 Comparative Analysis

This section details the comparison between the traditional classification approaches used in P2P applications (see Section 2.2). It then evaluates the available intelligent distributed classification algorithms (see Section 2.3), in regards to the server-independency, large data scalability and efficient online learning. In addition, this

section also evaluates the efficiency and effectiveness of the algorithms in dealing with imbalanced data distribution, dynamic networks, asynchronous system and large networks.

2.6.1 Comparison of Traditional Classification Approaches within P2P Networks

The classification processes in P2P networks are mainly performed through fully-distributed reputation-based approaches due to the absence of a server [94, 44, 92, 221, 51, 198, 222]. In these approaches, users manually give their own votes for items that they have seen before, and then the reputations for these items are calculated using aggregation techniques. The problem of reputation-based approaches is that they are unable to evaluate “never-before-seen” objects within the system or within the evaluation region. The reputations must be calculated for every item of interest, and they must be available within the query region. This requires a high number of communications among peers for the calculation of reputation.

Screening approaches [113, 90] require only a small number of samples and can predict a “never-before-seen” item. However, they require a clear understanding of the problem domain and sufficient information for people to build the rules. These rules may be difficult to build when the problem domain is complex and intangible. Extremely simple rules may result in inaccurate results while overly complicated rules may lead to over-fitting. The spam detection that was proposed by Jia [90], for instance, may not work for other types of spam (other than those which were considered in building the rules). This would require a frequent rule update, which involves rigorous data collection and analysis to manually determine rules for each update.

A straightforward method to perform classification of objects within P2P networks is the signature-based approach e.g., Donkeyfakes [58] and Sig2Dat [176]. Most of the signature-based approaches can only work in the presence of a server,

which is expensive to maintain and prone to a single point of failure. A signature-based approach in [220] offers a solution with a decentralised storage and it also can predict a “never-before-seen” object. However, it requires the storing of signatures (identifiers) of every object and considering a large number of objects that can be easily created and added into a P2P network, this consumes a high storage overhead. The large number objects may further results in a high communication overhead in its operation, since this requires the identifiers of all objects which have any feature matches the features of a queried object to be sent to a requesting peer in Zhou et al. [220].

Pattern recognition provides a sophisticated, effective, and efficient approach for classification. Pattern recognition is a promising approach in incorporating computational intelligence in the classification process specifically in dealing with large and complex information [62, 134, 105, 106, 85, 29]. It learns the regularities in data from training instances and predicts the class of an unknown object. Hence, it does not require to store every objects’ identifier within the system and able to make the classification of a “never-before-seen” object. As alternative to the traditional classification approaches, this thesis investigates the implementation of a pattern recognition for classification within P2P networks.

2.6.2 Task Distribution and Peer Involvement in DPR

In addition to the available P2P-based algorithms (i.e., P2P Bagging Cascade RSVM [8], PeDiT [17], MoM-NLDSVM [66], P2P Cascade RSVM [7], Distributed Boosting [108] and Ivote-DPV [125], we include other available distributed pattern recognition algorithms (i.e., Bagging [23], Boosting [169], HGN [140], GN [99] and BPNN) to compare their potential for implementation within P2P networks. In the analysis in this section, we classify the intelligent distributed classification algorithms into four categories based on their task distribution:

- (i) *semi-distributed* refers to an algorithm which requires a single host to coordinate or perform some parts of the computation while some parts of the computation are performed in distributed among peers.
- (ii) G_R -*distributed* refers to a cluster-based algorithm where a group of peers, G_R (which form a cluster) perform a fully-distributed computation in collaboration among peers within the group.
- (iii) *local algorithm* refers to a distributed algorithm where every peer builds its own global classifier or prediction by communicating with a small set of peers, that is Ω peers.
- (iv) *complete graph algorithm* refers to a distributed algorithm where every peer builds its own global classifier or prediction in collaboration with all other peers in the network.

The classification of distributed pattern recognition algorithms based on their task distribution is summarised in Table 2.1. The peer involvement can be cate-

Table 2.1: Category of distributed PR algorithms based on task distribution

Category	Examples
semi-distributed	Bagging [23] Boosting [169] HGN [140]
G_R -distributed	GN [99] BPNN
complete graph	P2P Cascade RSVM [7] Distributed Boosting [108]
local algorithm	Ivote-DPV [125] MoM-NLDSVM [66] PeDiT [17] P2P Bagging Cascade RSVM [8]

gorised as either cluster-based or P2P-based, where the cluster-based involvement requires a cluster formation, while the P2P-based involves all peers in the network and does not require any cluster formation. All G_R -distributed algorithms (e.g., NN and GN) are cluster-based while semi-distributed algorithms can either be cluster-based or P2P-based. Ensemble learning algorithms [23, 209, 67, 169] and the HGN [140] are cluster-based and they are semi-distributed—a coordinator node

is required to partially process the computation or coordinate the process. The outputs from the central node are broadcast to all peers within the cluster, or the outputs from every peers within the cluster are collected at the central node.

The peer involvement in local algorithms and complete graph algorithms are P2P-based. In local algorithms (e.g., P2P Distributed ID3, MoM-NLDSVM, P2P Bagging Cascade RSVM, and Ivote-DPV), the knowledge is exchanged among peers in an epidemic manner to a point where they reach a consensus—local knowledge at every node converges to global knowledge. Let Ω be the number of neighbours of a peer then every peer computes its own local results by aggregating local results from their Ω neighbours. The P2P Cascade RSVM [7] and the distributed boosting [108] are complete graph algorithms—every peer has to exchange its local results with all other peers within the network.

2.6.3 Efficient Online Learning

The learning phase in pattern recognition usually consumes a large amount of computational time as it requires several iterations before convergence. For a large training data, the training time may take days or even weeks on serial implementation [183]. Therefore, a scalable distributed algorithm within P2P networks must be able to perform incremental learning (in an online mode) where the learning is performed per instance instead of per batch. This requires a fast processing time in local learning and model aggregation (particularly when involves a high number of local models) as the training data is frequently updated. Table 2.2 compares the distributed pattern recognition algorithms with respect to the learning mode, the learning time at the peer level (the computation at local classifiers) and the learning time at the network level (involve an aggregation process to learn a global classifier).

Most ensemble learning schemes [169, 23, 108], the P2P Bagging Cascade RSVM [8] and the P2P Cascade RSVM [7] adopt batch learning as the learning procedure is usually expensive and requires a large number of iterations. Since a small change in

Table 2.2: Comparison on efficient online learning

Classification Methods	Online/ Batch	Learning Time Peer Level	Learning Time Network Level
Bagging [23]	batch	*	non-iterative
Boosting [169]	batch	*	iterative
P2P Cascade RSVM [7]	batch	high	iterative
HGN [140]	online	non-iterative	non-iterative
GN [99]	online	non-iterative	non-iterative
BPNN	online	non-iterative	iterative
Distributed Boosting [108]	batch	*	iterative
Ivote-DPV [125]	batch	iterative	non-iterative
MoM-NLDSVM [66]	online	iterative	iterative
PeDiT [17]	online	non-iterative	iterative
P2P Bagging Cascade RSVM [8]	batch	iterative	iterative

* the computational complexity depends on base classifiers.

a dataset may lead a different classifier or change in predictions, it is more feasible to wait for a reasonably large change of datasets before rebuilding the classifiers. However, P2P systems involve a large training data and frequent data changes, therefore batch learning here becomes inefficient [164]. Therefore, online learning is adopted in Forero et al. [66], Bhaduri et al. [17], Nasution and Khan [140], Khan [99] to ensure the local classifiers are up-to-date. Although there is no communication among peers during learning in the Ivote-DPV [125], the Ivote algorithm which is used as local classifier is a batch learning algorithm. Hence, we classify it under the batch learning category here.

One of the challenges in performing online learning in a fully-distributed system is that the aggregation process to obtain a global model usually requires a highly-iterative process. A naive method may be considered, for example by performing online local learning at a peer, then peer then broadcasts its updated models or statistics to all other peers within the network. This method, however, is not scalable for large networks. Local algorithms on the other hand, involves aggregation methods which are fully-distributed and scalable for large networks. Nonetheless, these methods involve a highly iterative process which are intractable in dealing with rapid data updates. The high magnitude of iterations during aggregation, to obtain a global classifier/prediction involves a large number of communications between peers. This results in a significant delay of learning time.

In this thesis, we define an efficient online learning distributed algorithm as follows.

Definition 2.1 (efficient online learning). *A distributed algorithm performs efficient online learning when it:*

- (i) learns in online mode; and*
- (ii) has a low learning time at the peer level (at local classifiers); and*
- (iii) has a low learning time at the network level.*

The parallel implementation of BPNN builds a fully-distributed system and it can be designed to learn in an online learning mode [182, 187]. However, implementing it on a P2P network is difficult since it involves substantial communication activities due to the large number of learning iterations. Furthermore, the restricted synchronisations between nodes require an extremely fast interconnection. On the contrary, the GN and the HGN can also perform online learning by using significantly less communications costs since they perform single-cycle at the peer level and network level. These result in a fast and efficient in-network computation.

Ensemble classifiers usually can perform a local learning operation in parallel, except in boosting [169], where each classifier learns in sequential— involving a large number of iterations [153]. As a result, the learning procedure in boosting is usually very slow. The sequential-reduction is performed in bagging, P2P Cascade RSVM, P2P Bagging Cascade RSVM and distributed boosting [23, 7, 8, 108]. It may require many aggregation steps (which is an iterative process) particularly when dealing with a large number of local classifiers. The collection of all models from all these local classifiers results in a high communication overhead and a long processing time.

In most local algorithms [17, 66], every peer exchanges statistics or predictions with a small number of its neighbours. This requires message exchanges in several iterations until terminated by a predefined termination condition. For example, the distributed SVM [66] and PeDiTl [17] are proven to converge into a global classifier

with is equal to its centralised counterpart, or to a decision with performance equal to a centralised classifier after a large number of iterations^{2.9}. In the worst case, the number of iterations in these local algorithms to achieve the global consensus is bounded by the network size. Unlike the PeDiT and MoM-NLDSVM, the Ivote-DPV does not build a global classifier, but, it aggregates the local predictions into a global prediction during recall (classification) phase.

2.6.4 Lack of synchronisation

Considering the latency in P2P networks, the algorithm should have a lack of synchronisation to fit in the asynchronous network. The algorithms in Forero et al. [66], Bhaduri et al. [17], Ang et al. [8], Luo et al. [125], Lazarevic and Obradovic [108] do not require synchronisation. Therefore, they are suitable for the relaxed asynchronous P2P networks. Boosting [169] also does not require synchronisation while bagging [23] and stacking [209] do require a central node to reduce the computation. In Breiman [23], Wolpert [209], Ang et al. [7], every local classifier computes the local samples from batch t sampling simultaneously and passes the output t to the reducer. The reducer then performs averaging or voting for the batch t after receiving outputs from all classifiers. This may require some sort of synchronisation, but does not require strict synchronisation since the failure of receiving an output from any local classifier does not fail the whole system. In contrast, the BPNN, HGN and GN do require strict synchronisation for each training and prediction.

2.6.5 Invariant to imbalanced data distribution

Ang et al. [9] highlights the issues of imbalanced data distribution that includes non-i.i.d class data distribution and small local datasets for P2P classification. In P2P networks, data can be highly distributed. Some peers may have samples of one class while other peers may have samples of all classes; i.e., an imbalanced class

^{2.9}The experiment in Forero et al. [66] shows that with kernel's dimension $L = 800$ and network size $N = 25$, the proposed algorithm in Forero et al. [66] requires approximately 4,000 iterations before converging into centralised classifier.

distribution. It is also possible that some peers have a very small number of local samples which can be insufficient for learning. P2P data distribution depends on the overlay network mechanisms (structured and unstructured) used and domain applications. In file sharing systems within unstructured networks, for instance, the file distribution according to user interest follows the Zipf distribution [168]. Given this scenario, it is important to have a learning scheme which is invariant to the imbalanced data distribution.

Most of the available ensemble classifiers [169, 23, 7] are built on the assumption that the training examples have an i.i.d distribution, and the training subsets are randomly sampled from a known single dataset. Stacking [209] requires the dataset to be partitioned into several disjoint subsets. In Ang et al. [7], the sample for each local classifier has equal size and is assumed to be randomly sampled from a single dataset.

The splitting criteria functions in decision trees are shown to be sensitive to the effects of imbalanced class distribution [65]. While for a biased training data distribution^{2.10}, MoM-NLDSVM [66] cannot construct an approximate local classifier by itself. It was reported that the performance of some stable classifiers are slightly degraded in bagging [23] due to the use of small training sets [23, 14, 153]. This problem is encountered in the distributed boosting [108] by pre-defining the minimum sample size for training at every peer. In the experiments by Ang et al. [8] and Luo et al. [125], we noticed that the size of the local datasets was ensured to be large enough so that they were sufficient for local training. This implies that small local datasets are expected to degrade the accuracy in these algorithms (Ivote-DPV and P2P Bagging Cascade RSVM).

Compared to other approaches, GN [99], HGN [140] and BPNN are readily distributed algorithms. Each node is responsible for a part of the function in the algorithms and the nodes in these algorithms jointly build a single global model. Therefore, an imbalanced training dataset is not an issue.

^{2.10}For instance, a large number of training examples of class \mathcal{C}_1 and a small number of training examples of class \mathcal{C}_2

2.6.6 Scalability within Large Networks

Given a distributed pattern recognition system within a large network, observations at all local sites must be included to build a global model/prediction for exact global classification^{2.11}. A scalable algorithm for large networks is a distributed algorithm which its use of resources grows slowly with an increase in network size, while the communications at every node is independent of the network size. We formally define a network-scalable algorithm as follows.

Definition 2.2 (Network-scalable). *A distributed algorithm is network-scalable if the communication per peer is independent of the network size (N).*

Ensemble classifiers, NNs and GN-based algorithms are designed for a small network with a fixed, predefined topology. They are not scalable for large networks. Complete-graph algorithms, such as the P2P Cascade RSVM and the distributed boosting [108] are also not scalable for large networks where the communication cost per peer increases linearly with an increase in network size and the overall communication overhead also grows at a linear rate.

Local algorithms (e.g., Ivote-DPV, PeDiT, MoM-NLDSVM) have communication complexity per peer $\mathcal{O}(|\Omega| \cdot I)$ where I is the number of iterations and Ω is a set of its immediate neighbours [125, 17, 66]. Hence, they are independent of network size and scalable for large networks. The worst communication cost in local algorithms (when the disagreement among nodes is very high or the misclassification gain value is small) is $\mathcal{O}(N)$ iterations [17]. However, they normally avoid unnecessary communications. A node sends messages its Ω neighbours only when a change to the statistics in the local dataset is beyond a predefined condition threshold. The global SVM model is built in Forero et al. [66] by exchanging information among Ω neighbours in several iterations until reaching the termination condition. While the

^{2.11}An exact global classification equals to the centralised classification which is obtained when the whole data within a network is located at a single site.

P2P Bagging Cascade RSVM adopts a stochastic approach that requires aggregation among random k and j peers during both learning and prediction respectively given that $k, j \ll N$.

2.6.7 Scalability for Large Datasets

In order to build a scalable algorithm for large datasets within a P2P network, the algorithm needs to be communication-efficient and resource-efficient in dealing with a large number of training data and a high-dimensional dataset (a dataset with a large number of attributes). Since the computation is distributed among peers, the computation needs to be decomposable into components which each does not require an expensive computation. This is important to encourage peer participation in the system. This thesis measures the large datasets scalability of a distributed pattern recognition algorithm based on its resource efficiency per peer and its communication efficiency for large datasets.

The methods in Breiman [23], Schapire et al. [169], Lazarevic and Obradovic [108], Forero et al. [66], Luo et al. [125], Ang et al. [7, 8] are considered as communication-efficient since the overall communication overheads in these algorithms are either independent of or at most linear with the size of the training data and the number of attributes. Excluding the HGN [140], GN [99, 101] and BPNN, other methods require the whole vector of a training instance to be at a single location. As a result, the whole pattern recognition computation has to be performed on a single peer. Hence, the computation complexity per peer is not decomposable and is expensive. Although the size of the training dataset at each base classifier is small, the computational cost per node is still high given the fact that most of the available state-of-the-art algorithms are computationally intensive. The learning complexity at each node remains high with several iterations required for model convergence. This limits the scalability of the distributed pattern recognition in dealing with large data. The computation load can burden low-resources peers (e.g., smartphones and tablets) and this is worse when dealing with a high-dimensional problem.

For instance, a SVM-based local classifier [7, 66] requires high computational complexity in solving a quadratic optimisation problem. Given n as the size of the local training data at a peer, the size of the subset of the dataset used in RVSM [7] (l) is smaller than n . The resultant time complexity of RVSM is $\mathcal{O}(l^2)$ which is smaller than the traditional SVM ($\mathcal{O}(n^3)$), but at the cost of degradation on accuracy of local classifiers [119]. SVM is designed for binary classification, but can be extended to multi-class classification problems by using one-versus-one or one-versus-all classifier. This means several models have to be built for each pair of classes where, in solving c -class problem using one-versus-one, $|\mathcal{C}| = \frac{c(c-1)}{2}$ pairwise classifiers are required in total. In the Ivote-DPV [125], each peer performs the whole Ivote learning algorithm on its local data and, as a result, the local learning time complexity at per peer is high considering Ivote learning requires several iterations.

The implementation of NN algorithms on a single PC is inefficient as they require a high computational complexity particularly when dealing with high-dimensional problems [85, 27, 151]. By having a fully-distributed neural network algorithm on a network of processors, the expensive computation is distributed efficiently into a lightweight computation per processor. The high dimension, however, involves a large number of neurons and adjustable weights in BPNN [151]. For instance, a one layer BPNN may have $\frac{d+c}{2}$ hidden neurons which means the adjustable weight is $\mathcal{O}(d^2)$. This consumes a high communication cost and, given a large number of iterations involved, it requires a high connectivity with significant communication speed between peers which cannot be guaranteed in P2P networks.

As the number of processing units strictly depends on the problem dimension, a high-dimensional distributed pattern recognition can not be easily implemented when insufficient number of peers available within the network. Similarly in the HGN, a large number of nodes ($\mathcal{O}(u \cdot d^2)$ where u is the domain size) is required when dealing with large dimensional problems. Therefore, BPNN, and the HGN can be considered as resource-efficient per peer but not communication-efficient. However,

in the GN architecture, the number of nodes that are required is $\mathcal{O}(u \cdot d)$ —it can be considered as resource-efficient per peer and communication-efficient.

In PeDiT [17], the base classifier^{2.12} computation requires $\mathcal{O}(n \times d^2)$ training time and $\mathcal{O}(d^2)$ classification time. The number of condition checks are d^2 . Thus, the communication cost for each iteration is $\mathcal{O}(d^2 \cdot |\Omega|)$ per peer where Ω is the number of neighbours of every peer. On top of that, the number of majority voting instances per attribute pair increases exponentially with an increase in the number of distinct attribute values [17]. This results in an expensive communication cost given that each condition checked in the DMV may require $\mathcal{O}(N)$ iterations at the worst case. As the training data in P2P networks is frequently updated, the distributed pattern recognition system requires frequent network retraining. A high number of iterations involves frequent information exchanges which contributes to a high magnitude of communication. These show that the PeDiT algorithm [17] is neither resource-efficient per peer nor communication-efficient, inferring the low scalability of this approach in handling high dimensional data.

2.6.8 Fault-tolerance

Mapping an algorithm on a dynamic network requires a fault-tolerance approach to ensure a constant availability and accuracy of the prediction. Therefore, it is important to ensure the algorithm is fault-tolerant in dealing with the dynamic network. Distributed pattern recognition algorithms in Forero et al. [66], Bhaduri et al. [17], Ang et al. [8, 7], Luo et al. [125], Lazarevic and Obradovic [108] are naturally more robust to the network churn as the computations are local at each peer. The local decision and local model can be adapted smoothly to the changes of neighbourhood or the network topology.

^{2.12}ID3 algorithm is used as the base classifier in in Bhaduri et al. [17]

All cluster-based algorithms including ensemble learning schemes, GN-based algorithms and NN algorithms assume a stable distributed computing platform. However, this is not the case in the dynamic P2P networks. The ensemble based approaches [169, 23, 209, 67, 108] have loosely-coupled processes where any failure of the processes does not fail the entire computation. Instead, the failure degrades the system performance and slows down the convergence time. In contrast, the inter-node connections in GN-based algorithms are tightly-coupled where each node stores indices from its immediate neighbours. Thus, the failure of any neighbour of a node p in GN-based algorithms will cause the stored memory in node p to be incorrect and the failed node cannot be easily replaced with any other nodes. Although BPNN also has tightly-coupled processes, each node at each layer operates independently. Given the large number of nodes and the nature of distributed computations in BPNN, failure of any node will not fail the overall computation.

2.7 Summary

This chapter defines the impetus for studying a scalable distributed pattern recognition that works efficiently and effectively with large data, dynamic, asynchronous, imbalanced data distribution and large P2P networks. Firstly, P2P technology, architectures and applications were discussed. The concept and theories of distributed pattern recognition were presented and different kinds of approaches to these were discussed. In this research, we are particularly interested in designing a scalable distributed pattern recognition system for P2P networks. Four strategies to achieve this is by devising a server-independent, large dataset scalable, efficient online learning and large network scalable algorithm.

Unlike other distributed computing platforms, like grid computing or multi-core computing, P2P networks are usually chaotic and have a dynamic membership. Moreover, the link delay in P2P networks depends on the speed of Internet connection, the available bandwidth and the network size. Thus, a distributed algorithm within P2P networks must be flexible in terms of system synchronisation. Attaining

another characteristic would lead to a better distributed pattern recognition algorithm within P2P networks that is, the invariant to imbalanced data distribution: imbalanced class distribution and small local datasets. The advantages and limitations of each approach in relation to computational scalability for large datasets, dealing with dynamic and asynchronous networks, imbalanced data distribution and large networks issues were discussed. The comparative analysis of the available distributed pattern recognition here lays the foundation on the direction of this thesis. This is summarised in Table 2.3, which shows that the available methods do not encounter all of these issues. P2P networks are fully-distributed systems. Therefore, an application to be implemented on top of a P2P overlay must also be fully-distributed. The second issue in most of the distributed pattern recognition algorithms is that the whole pattern recognition computation (usually expensive) is performed on a single peer. This, in turn, limits the ability of these algorithms to scale and compute a high-dimensional dataset. Since peers in P2P networks are volunteers, who expect to contribute a small part of their resources, the high computational cost may discourage the peer participation.

The available local algorithms are highly iterative for convergence into a global classifier, making it inefficient for P2P networks which have no guarantee of fast connection between peers. Some of these algorithms perform an online learning mode to cope with the frequent data updates [66, 17]. Although incremental learning algorithms such as k -NN and naive Bayes can be used at local classifiers, the frequent global model rebuilding is very expensive considering the high iterative process per learning. The Ivote-DPV, however, does not build a global classifier but builds a global prediction by using DPV which is also a highly iterative algorithm for large networks. In the Ivote-DPV, the Ivote model is used as local classifiers. Although an online learning is possible in Ivote, it is intractable as its local computation requires several learning iterations per learning. Hence it is inefficient for online learning.

The only algorithms which distribute their computation into a number of fine-grained components are BPNN and the GN [99]. In fact, the GN is also the only

Table 2.3: Comparison on distributed pattern recognition approaches

Classification Methods	Resource Efficient	Communication Efficient	Server-independent	Efficient online learning	Lack synchronisation	Fault tolerant	Network-scalable	Invariant to imbalanced data distribution
Bagging [23]	*	yes	no	no	yes	yes	no	no
Boosting [169]	*	yes	no	no	yes	yes	no	no
P2P Cascade RSVM [7]	no	yes	no	no	yes	yes	no	no
HGN [140]	yes	no	no	yes	no	no	no	yes
GN [99]	yes	yes	yes	yes	no	no	no	yes
BPNN	yes	no	yes	no	no	yes	no	yes
Distributed Boosting [108]	*	yes	no	no	yes	yes	no	no
Ivote-DPV [125]	*	yes	yes	no	yes	yes	yes	no
MoM-NLDSVM [66]	no	yes	yes	no	yes	yes	yes	no
PeDiT [17]	no	no	yes	no	yes	yes	yes	no
P2P Bagging Cascade RSVM [8]	no	yes	yes	no	yes	yes	yes	no
* the computational complexity depends on base classifiers.								

fully-distributed algorithms which performs efficient online learning. It performs a single-cycle learning as a local classifier and also a single-cycle learning during aggregation to form a global classifier. This speeds the computation time for each training pattern. Given the nature of P2P networks with rapid data updates, this is an appealing characteristic. In addition, unlike other methods which usually are affected by the imbalanced class distribution and small training examples at local classifiers, GN is invariant to imbalanced data distribution. These encourages us to use the GN model in the process of devising a fully-distributed pattern recognition algorithm that can successfully be implemented within P2P networks.

GN requires a significantly less number of nodes compared to HGN. However, as noted by Nasution and Khan [140], the algorithm has an accuracy issue. In addition, it requires synchronisation and it also does not properly deal with dynamic networks. The fact that the GN is a cluster-based algorithm, means that it does not encounter the network size scalability issue. It should be noted that these issues are solved in local algorithms [125, 17, 66, 8].

In the next chapter, we present a fully-distributed pattern recognition algorithm that is scalable for large data. The local algorithms usually give approximate prediction and require at least an $\mathcal{O}(N)$ communication cost to obtain an exact prediction^{2.13}. Hence, an algorithm that can give exact predictions and have an overall communication overhead that is independent of the total number of peers, is desirable. This algorithm is proposed in the following chapter and there, we also show how the proposed method deals with an imbalanced data distribution. This followed by Chapter 4, where we show how the proposed method deals with large datasets and dynamic networks.

^{2.13}An approximate prediction is made by considering a subset of global training samples, while an exact prediction would include all global training samples.

Chapter 3

P2P-GN for Distributed Pattern Recognition within P2P Networks

P2P networks are usually large-scale where they usually involve hundreds of thousands to millions of peers, and these peers are self-organised^{3.1}. These peers collect their own data independently and the distributed pattern recognition (PR) problem in this network is to build a classification solution^{3.2} that represents the global dataset^{3.3}. The most common way is to send all the training data to be processed at a single site—i.e., centralised processing. This, however, may result in the site becoming a bottleneck, in particular when involving a large amount of data. Therefore, either fully-distributed processing or semi-distributed processing is used. In this work, we focus on the fully-distributed processing to suit the computing environment of P2P networks. In the absence of a centralised node, the question is, how to integrate the data from all nodes in the network so that the resulting global model/prediction represents the whole network—producing an exact global classification?

Different fully-distributed classification algorithms have different ways of how the learning or prediction converges to a global classification solution. The performance of these algorithms depend on the magnitude of inter-connection between

^{3.1}The peers in P2P networks can leave and join the networks automatically.

^{3.2}Model or prediction.

^{3.3}The whole dataset across the network.

peers, the number of iterations and the degree of simultaneous processing that takes place. Although the classification algorithms that are built based on the Distributed Majority Voting (DMV) [207] and Distributed Plurality Voting (DPV) [125] have been effective for large-scale distributed P2P networks, they require several iterations that is bounded by the number of local classifiers to produce an exact global classification. This may delay the classification process, or convergence may take too long for complex problems (e.g. high number of attributes) or hard to classify problems (problems with many conflicting decisions) [125, 17, 66]. The large convergence time involves a high number communication exchanged between peers. In addition, the high number of peers may increase conflicts in decision making which delay the convergence, and this results in high communication overhead.

There are classification algorithms for large-scale distributed system which only require a single-cycle learning. However, they either require every peer to collect local models/predictions from all other peers (complete graph classification algorithms), or randomly select peers to aggregate the data. Complete graph algorithms such as P2P Cascade RSVM [7] and distributed boosting [108] incur a quadratic communication overhead that is not feasible for large networks. Randomised algorithms such as P2P Bagging Cascade RSVM [8], however, produce approximate classifications, which all peers might not agreed to, and may lead to a less accurate solution. Our focus in this thesis, however, is to build an accurate exact global classification algorithm that represents the global dataset^{3.4}.

In addition, most of the available distributed algorithms assume a uniform data distribution which is not always the case in the real world P2P environment [125, 66, 108, 7]. Users usually have different interests and social interactions. Consequently, the type of data or files which serve as the primary source of the local data are naturally distributed in a non-uniform way. The real world observations from several studies [168, 49] show that the data distribution in the P2P-based file sharing is usually imbalanced, in terms of type and volume of data shared. It is observed that

^{3.4}Dataset which represents the whole network.

peers in a P2P network are, in general, interested in a subset of the total available content on the network where these peers are often only interested in files from a few content categories [49]. World Wide Web (WWW) exhibits Zipf distributions on the popularity of documents [168]. This reflects the fact that some popular documents are very widely copied and held, while most documents are held by far fewer peers. Hence, our aim in this study is to bring on an efficient algorithm that can produce an exact global classification in a single-cycle and invariant to the imbalanced data distribution. The importance of having this characteristics for P2P-based PR algorithms have been discussed in Chapter 2. Unlike other algorithms, which build many global classifiers^{3,5}, we are building a single global classifier within P2P networks.

The work in this chapter is developed based on the guidelines as discussed in Chapter 2. The Graph Neuron (GN) [99] is a single-cycle distributed algorithm with high parallelism. This characteristic attracts us to use the algorithm in this thesis, since it can encourage a fast learning and classification process with low communication overhead. The lightweight processes at every node are also beneficial here since we also consider the resource-awareness factor. This is due to the trends today which show P2P networks also consist of resource-constrained devices such as tablet PCs or smartphones. Another reason to use the GN is that, it is not only readily distributed; but it can further be easily distributed into many fine-granularity processes which suits our strategy in this work.

Considering the reasons above, the GN is the best algorithm to use as a foundation to achieve our aim in this study. The feasibility of GN in this work however, is subject to several improvements. This includes to solving its accuracy issue; and making it fully-distributed and asynchronous. In order to make it work effectively in the fully-distributed and asynchronous architecture, we propose a two stage prediction approach, which is a hybrid voting and weighting, to support the decision making. The processes in the modified algorithm are then further divided into

^{3,5}Every peer builds its own global classifier/prediction through distributed aggregations.

many fine-granularity processes which then are arranged in the network using the Distributed Hash Table (DHT) [6] system. This makes the algorithm adaptive to the increasing or decreasing network size, which results in a scalable approach for large-scale distributed networks. We call the new resulting algorithm, the P2P-GN.

Our focus in this research is to produce a global prediction during classification instead of during learning. This is feasible considering the large amount of data and the frequent data update in the P2P environment may require rapid learning process to ensure the classifier is up-to-date. Hence, the closest approach to our work is the Ivote-DPV [125] which mainly used to compute its performance with our algorithm.

The chapter starts by introducing the P2P-GN algorithm for PR within P2P network in Section 3.1. This section describes the logical structure and memory structure of the algorithm which involves fine-grained and loosely-coupled memory units. Then, we briefly describe the placement of these memory units over a structured P2P network. This is followed by the description of the learning and recall (classification) procedure.

In Section 3.2, we present the results from a series of experiments in evaluating the P2P-GN. We initially show that the P2P-GN has comparable accuracy with six state-of-the-art algorithms (backpropagation neural network (BPNN), radial basis function network (RBFN), k -nearest neighbour (k -NN), nearest neighbour (1-NN), decision tree ID3 and naive Bayes) on centralised implementation in several experiments using publicly available datasets. Then, we show that it is efficient (communication and time) compared to the Ivote-DPV and ensemble ID3. We also demonstrate the invariant of the algorithm for imbalanced data distribution compared to Ivote-DPV, ensemble ID3, ensemble k -NN, and ensemble naive Bayes in this section. Additionally, we demonstrate that the workload distribution in the P2P-GN is adaptive to various network sizes and this achieved by using DHT systems instead of cluster-based implementation.

In the following Section 3.3, we theoretically analyse the complexity of our algorithm and compare them with ensemble k -NN, ensemble ID3, P2P-Cascade SVM,

P2P-boosting and Ivote-DPV. Finally, we summarise this chapter in the conclusion in Section 3.4. Note that the notations that are used in this chapter can be referred in Tables A.1 of Appendix A.

3.1 Overview of P2P-GN

This section describes our proposed algorithm that is the P2P-GN. The main goal of this algorithm is to produce an exact classification that is accurate and invariant to the imbalanced data distribution within large-scale distributed networks, whilst minimising the run-time and communication overheads. The algorithm has the following characteristics: fully-distributed, lightweight, online, asynchronous and single-cycle. This characteristic makes it feasible for large networks. In this algorithm, the whole network forms a global associative memory that collaboratively memorises incoming patterns and predicts the objects' classes. This operation is fully-distributed where classification queries can be submitted and predictions can be made from any node within the network.

3.1.1 The Input of P2P-GN

Let $\mathcal{F} = \{f_1, f_2, \dots, f_d\}$ be a set of features (neglecting the orders) and $\mathcal{SF} = \{sf_1, sf_2, \dots, sf_d\}$ be a sequence (orderly-arranged) of \mathcal{F} . For spatial data, the features can be arranged in the order of location, while, for temporal data, the features can be arranged in the order of time. A sequence of feature vector $\mathbf{x} = \{x_1, x_2, \dots, x_d\}$ of an instance \mathbf{x}_A consists of a sequence of values associated with features \mathcal{SF} which describe the instance \mathbf{x}_A where x_i is the value of sf_i and i is a feature position within \mathcal{SF} .

A classification task is to distinguish a test instance A (which is represented by \mathbf{x}) into a class $c \in \mathcal{C}$. Let $\{\langle \mathbf{x}_j, c_j \rangle\}_{j=1}^n$ denotes a training dataset for a classification problem where n is the number of training instances, $\langle \mathbf{x}_i, c_i \rangle$ is a pair of feature vector and class label for i th instance, respectively. During learning, an input instance is a pair $\langle \mathbf{x}, c \rangle$, while during recall (classification), an input instance is \mathbf{x} since we are

predicting c . An example of \mathcal{F} , \mathcal{SF} and \mathcal{C} are given in the following by using the car evaluation problem [13].

$$\begin{aligned}\mathcal{C} &= \{\text{unacceptable, acceptable, good, very good}\} \\ \mathcal{F} &= \{\text{buying, persons, maintenance, safety, doors, lug boot}\} \\ \mathcal{SF} &= \{\text{buying, maintenance, doors, persons, lug boot, safety}\}\end{aligned}$$

An example of a training instance \mathbf{x} in this problem is as follows.

$$\begin{aligned}c &= \text{unacceptable} \\ \mathbf{x} &= \{\text{high, high, 2, 4, medium, low}\} \\ \text{where} \\ \mathbf{x} &= \{\text{buying} = \text{high, maintenance} = \text{high, doors} = 2, \text{person} = 4, \\ &\quad \text{lug boot} = \text{medium, safety} = \text{low}\}\end{aligned}$$

An instance \mathbf{x} of class “unacceptable” in this example is presented by the pattern $\{\text{high, high, 2, 4, medium, low}\}$ where the i th element in the pattern refers to the i th feature in \mathcal{SF} . Note that we use the term pattern alternatively with feature vectors and the term sub-pattern alternatively with feature sub-vectors in this thesis.

Let \hat{x}_i be an i th sub-pattern of \mathbf{x} and \widehat{sf}_i be a sequence of features that are represented by \hat{x}_i . In other words, \widehat{sf}_i refers to a sequence of feature labels, while, \hat{x}_i refers to a sequence of feature values. The combined memory of all segments $\widehat{X} = \{\hat{x}_i\}_{i=1}^{n_H}$ constructs a memory of \mathbf{x} . A sub-pattern \hat{x}_i overlaps with a sub-pattern $\hat{x}_{(i+1)}$, with overlap rate OV where the first OV digits/positions of $\hat{x}_{(i+1)}$ are also the last OV digits/positions \hat{x}_i . The overlapping strategy improves the accuracy of the prediction. For instance, let \hat{x}_a overlaps with \hat{x}_b and \hat{x}_b overlaps with \hat{x}_c . This is effectively equivalent to the GN where each node in GN asks for values from its left and right neighbours. However, in this approach, the left and right values are directly given to the node; reducing the synchronisation process.

Examples of sub-patterns created from $\mathbf{x} = \{high, high, 2, 4, medium, low\}$ with $\mathcal{SF} = \{buying, maintenance, doors, person, lug\ boot, safety\}$ with $d_s = 3$ and $OV = 2$ is given in Table 3.1, while, with $d_s = 3$ and $OV = 1$ is given in Table 3.2. A large OV creates a large number of sub-patterns. In these examples, when OV is equal to two, four sub-patterns are created from \mathbf{x} , while, when OV is equal to one, three sub-patterns are created. In the example in Table 3.1, the sub-pattern \hat{x}_1 is overlapped with \hat{x}_2 at two positions $\{high, 2\}$ where $\hat{x}_1 = \{high, high, 2\}$ and $\hat{x}_2 = \{high, 2, 4\}$. In Table 3.2, the sub-pattern \hat{x}_1 is overlapped with \hat{x}_2 at one position $\{2\}$ where $\hat{x}_1 = \{high, high, 2\}$ and $\hat{x}_2 = \{2, 4, medium\}$.

Table 3.1: Example of segments generated from $\mathbf{x} = \{high, high, 2, 4, medium, low\}$ with $\mathcal{SF} = \{buying, maintenance, doors, person, lug\ boot, safety\}$, $d_s = 3$ and $OV = 2$.

SF	Input segments
$\widehat{sf}_1 = \{buying, maintenance, doors\}$	$\hat{x}_1 = \{high, high, 2\}$
$\widehat{sf}_2 = \{maintenance, doors, person\}$	$\hat{x}_2 = \{high, 2, 4\}$
$\widehat{sf}_3 = \{doors, person, lug\ boot\}$	$\hat{x}_3 = \{2, 4, medium\}$
$\widehat{sf}_4 = \{person, lug\ boot, safety\}$	$\hat{x}_4 = \{4, medium, low\}$

Table 3.2: Example of segments generated from $\mathbf{x} = \{high, high, 2, 4, medium, low\}$ with $\mathcal{SF} = \{buying, maintenance, doors, person, lug\ boot, safety\}$, $d_s = 3$ and $OV = 1$. The padding # is added so that the sub-pattern size is maintained $d_s = 3$ for every sub-pattern.

SF	Input segments
$\widehat{sf}_1 = \{buying, maintenance, doors\}$	$\hat{x}_1 = \{high, high, 2\}$
$\widehat{sf}_2 = \{doors, person, lug\ boot\}$	$\hat{x}_2 = \{2, 4, medium\}$
$\widehat{sf}_3 = \{lug\ boot, safety, \#\}$	$\hat{x}_3 = \{medium, low, \#\}$

As shown in the last column in Table 3.2, a padding value # is added at the last position in \hat{x}_3 . This preserves the size of the sub-pattern to $d_s = 3$, since there is no more value to include (*safety* is the last feature in \mathcal{SF}). The algorithm to generate the sub-patterns from an input pattern is described in Algorithm B.1 of Appendix B. The resulting number of sub-patterns, n_H is given in Equation 3.1.

$$n_H = \left\lceil \frac{d-d_s}{d_s-OV} + 1 \right\rceil \quad (3.1)$$

subject to $d > d_s > OV$ and $d, d_s > 0$

3.1.2 Logical Structure of P2P-GN

A prediction is made by associating the retrieved memory from leaf nodes at the requester. An input to a leaf node b_i is a sub-pattern \hat{x}_i which is a part of \mathbf{x} . The decision on what features are included in every sub-pattern is predefined. The logical structure of the P2P-GN consists of a *requester* and a number of *leaf nodes* that store the parts of the memory (sub-pattern).

Definition 3.1 (requester). *A requester is any node within the network which sends a learning and recall request to the P2P-GN system.*

Definition 3.2 (leaf node). *Given a segment \hat{x}_i , a leaf node is a node which responsible to store or retrieve \hat{x}_i .*

Every node in the network can be a requester and so the algorithm is fully-distributed. As shown in Figure 3.1, the logical structure in this algorithm is very simple, where every leaf node has a connection to the requester node. Note that this is a the view from a single peer which does show the whole system. Figure 3.1 shows

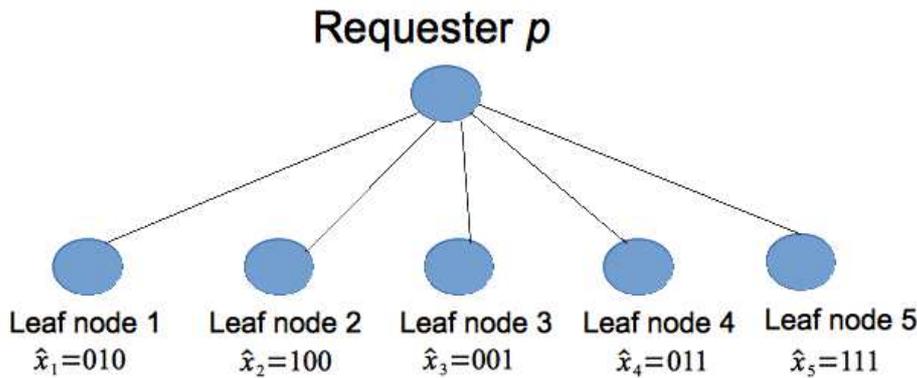


Figure 3.1: An example of logical P2P-GN structure view from a single node—requester. The five leaf nodes {leaf node 1, leaf node 2, leaf node 3, leaf node 4, and leaf node 5} are connected to the requester node; a leaf node i stores a disjoint segment \hat{x}_i .

an example of a P2P-GN structure for binary pattern: “0100111” (refer to Figure 2.4 and Figure 2.5 of Chapter 2 to see the structures of the equivalent pattern by using

the GN and HGN, respectively). In this example, a leaf node 1 stores $\hat{x}_1 = \text{“010”}$, a leaf node 2 stores $\hat{x}_2 = \text{“100”}$, a leaf node 3 stores $\hat{x}_3 = \text{“001”}$, a leaf node 4 stores $\hat{x}_4 = \text{“011”}$, and a leaf node 5 stores $\hat{x}_5 = \text{“111”}$.

P2P-GN has a logical structure where a leaf node b_i does not refer to a specific peer i within a P2P network (except in the cluster-based implementation where every leaf node is assigned to a specific peer). In the network-wide implementation^{3,6}, the physical locations of leaf nodes may vary among different patterns; only similar patterns have the leaf nodes which located at similar locations within a P2P network. In other words, the physical location (within a P2P network) of a leaf node i for a pattern \mathbf{x}_a does not guarantee the physical location of a leaf node i for a pattern \mathbf{x}_b . For instance, given a pattern $\mathbf{x}_a = \text{“10010”}$ and $\mathbf{x}_b = \text{“11100”}$. Let \hat{x}_1 be a sub-pattern of the first three position of a pattern. Hence, $\hat{x}_{a,1} = \text{“100”}$ and $\hat{x}_{b,1} = \text{“111”}$ where $\hat{x}_{y,i}$ refers to i th sub-pattern of a pattern y . The logical location of $\hat{x}_{a,1}$ and $\hat{x}_{b,1}$ are at a leaf node b_1 since they are the first subpattern of \mathbf{x}_a and \mathbf{x}_b , respectively. However, their physical placement within a P2P network maybe different— $\hat{x}_{a,1}$ maybe stored at the peer p_{34} , while, $\hat{x}_{b,1}$ maybe stored at the peer p_2 . Further explanation regarding the P2P-GN storage within P2P networks will be discussed in the next chapters.

The basic procedures in the P2P-GN operation are as follows. Given n_H sub-patterns with size d_s each, are arranged in order, $\widehat{X} = \{\hat{x}_1, \hat{x}_2, \dots, \hat{x}_{n_H}\}$. Each of the segment \hat{x}_i is processed locally by a leaf node b_i . The selection of sub-pattern size d_s must be made carefully before assigning jobs to peers. Too small d_s may result in a large number of leaf nodes which consumes a large communication costs, the requester peer that is responsible to aggregate the result will be burdened with too many received transactions to process. On the other hand, too large d_s may deteriorate the accuracy as it reduces the scheme capability to deal with noisy data. Thus, a cross-validation based on a small set of trial data can be performed to estimate a suitable parameter based on the available resources and the classification

^{3,6}An unstructured system with dynamic architecture that expand and shrink depending on the network

problem. To simplify this, we define the rule-of-thumb on d_s and OV which will be used throughout this thesis as follows.

Rule 3.3 (P2P-GN Rule-of-Thumb). *Given a pattern \mathbf{x} with size- d , the value d_s are defined to be $2 \leq d_s \leq \lfloor \log_2 d \rfloor$ with OV equal to $\lceil \frac{d_s}{2} \rceil$ where $d > d_s$.*

3.1.3 Distributed Storage

The memory in this algorithm is stored in a structure called a *bias element* which stores a pattern and the weight of labels (classes which are associated with the pattern in previous experiences). A sub-pattern \hat{x} is stored in a form of identifier ψ that is obtained by mapping \hat{x} into a \mathcal{R} identifier space by using an identifier generation function $\mathbf{q}(\cdot)$ (see Section 3.1.5 for details).

A bias element is basically an entry of a bias array (storage structure in GN) with additional storage called a *score table*. Figure 3.2 shows an example of how the table structure of a bias array is divided into several bias elements and these bias elements are linked to the score table which stores the weight for labels. In contrast to GN, a node p in P2P-GN does not store its neighbours' memory and this reduces dependency on its neighbours. Any changes to its neighbours' memory therefore does not affect the memory at the node p .

Definition 3.4 (bias element (ψ)). *A bias element ρ is a tuple $\langle \psi, \text{score table} \rangle$ where ψ is the sub-pattern identifier.*

Definition 3.5 (score table (ψ)). *A score table stores a list of pairs $\langle c, \hat{f}(\psi, c) \rangle$. A class label is represented by c and the $\hat{f}(\psi, c)$ is the frequency of ψ occurred as the class c .*

Repeating patterns are only stored during their first occurrence. Thus, the number of stored segments is also much smaller than the total number of globally stored patterns. Each time a new pattern is presented into the system, there is a possibility that some segments of the pattern have already occurred in previous training.

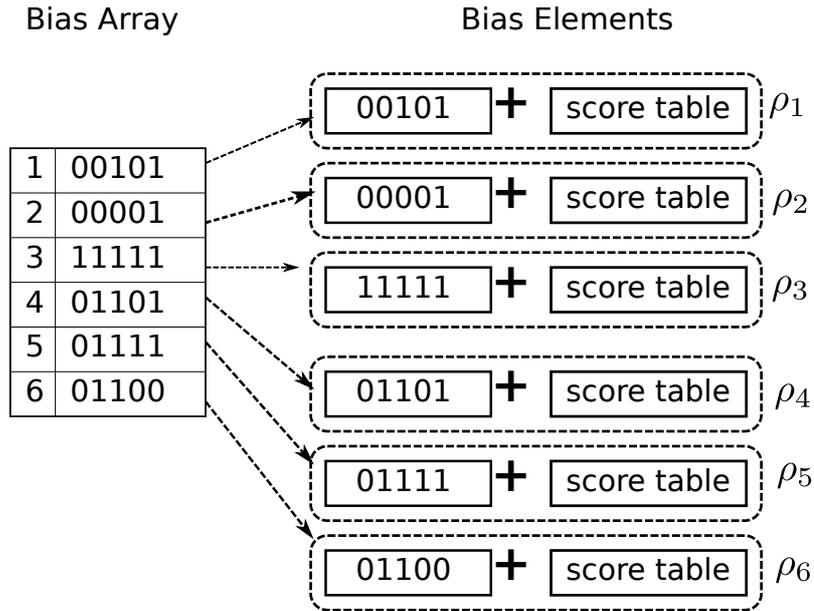


Figure 3.2: A bias element is a fine-granularity storage unit of a bias array in GN with score table. In this example, six bias elements $\{\rho_1, \rho_2, \rho_3, \rho_4, \rho_5, \rho_6\}$ are created from a bias array.

Gence, not all of their segments are stored as a new pattern (as shown in Figure 3.3).

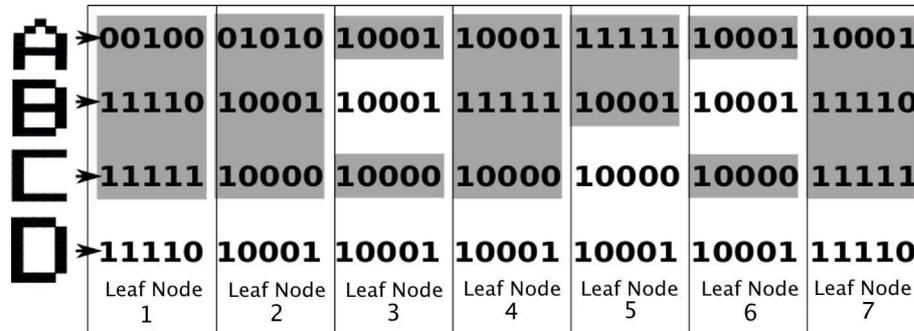


Figure 3.3: The stored entry(s) in bias array(s) for five 5×7 -size binary images at seven leaf nodes. The grey-coloured area shows the stored bias elements. There is no entry added for pattern “D” at any leaf node as all segments have been memorised previously.

3.1.4 Two Stage Prediction

In P2P-GN, we use two factors—vote and weight—to make predictions. In the first stage, the majority voting is used by the requester to select the candidate options

with highest vote. In the second stage, the weighted sum is used whenever there are several selected candidates (from the first stage) with the same majority votes. Leaf nodes poll a vote for the option(s) which is associated with a segment \hat{x}_a , and they also calculate the weight for each option. Given that a bias element only has a limited knowledge of sample distribution, then weight $w(c|\hat{x}_a)$ is calculated by dividing the frequency of the segment \hat{x}_a is occurred as c with the frequency of the segment \hat{x}_a occurrences. The imbalanced class distribution might severely result in incorrect decision if the weight is used alone—neglecting votes—in a final decision where the option with higher samples may dominate the result. However, the use of the voting approach solves this, by making it possible that the option c_b with small samples is selected as the potential winner if most of the leaf nodes vote for it, and the weight metric is no longer applicable if c_b is the option with the highest vote. The attempt of using only the voting approach is also ineffective, as it results in having many indecisive predictions—prediction with more than one decision of classes. The detailed description of this two stage prediction will be discussed in Section 3.1.8.

3.1.5 Bias Element Identifier Generation

In order to allow an efficient access throughout the system’s storage, a bias element of sub-pattern \hat{x} is mapped into a \mathcal{R} identifier space to create a *bias identifier* by using an *identifier generation* function $\mathbf{q}(\cdot)$. A bias identifier and an identifier generation process are formally defined as below.

Definition 3.6 (bias identifier). *A bias identifier ψ is a unique reference address of a bias element (ψ) in \mathcal{R} identifier space.*

Definition 3.7 (identifier generation). *An identifier generation is a process to obtain a bias identifier ψ of a bias element within \mathcal{R} identifier space using $\mathbf{q}(\cdot)$ function.*

$$\mathbf{q}(\hat{x}) : \hat{x} \rightarrow \psi \text{ where } \psi \in \mathcal{R}$$

An input argument to the $q(\cdot)$ function (which produces an identifier for an entry at a leaf node i) is either a raw sub-pattern \hat{x}_i or the combination of \hat{x}_i and its position in the feature space where \hat{x}_i is a set of small subsets from a pattern \mathbf{x} and i refers to its position. The function $q(\cdot)$ creates an identifier for \hat{x}_i , and one way to achieve this is by using a hash function to transform \hat{x}_i into \mathcal{R} space. Given \hat{x} has a length d and a binary domain $\{0, 1\}^d$, the identifier space has to be larger than d^2 to avoid collision. When the sub-patterns at the lower level are small enough in size, a trivial hashing can be used where the sub-pattern itself becomes its identifier. To get a minimal length of the identifier values, the \mathcal{R} space should not be too large. Therefore the function $q(\cdot)$ is chosen so that $2^{l-1} < \hat{a} < 2^l$ where \hat{a} is the number of possible pattern combination and l is the length of binary string which is large enough to avoid any possible collision. For instance, for a pattern with $\{A, B, C\}^5$, the number of possible combinations is $3^5 = 324$, thus $324 < |\mathcal{R}| < 2^9$.

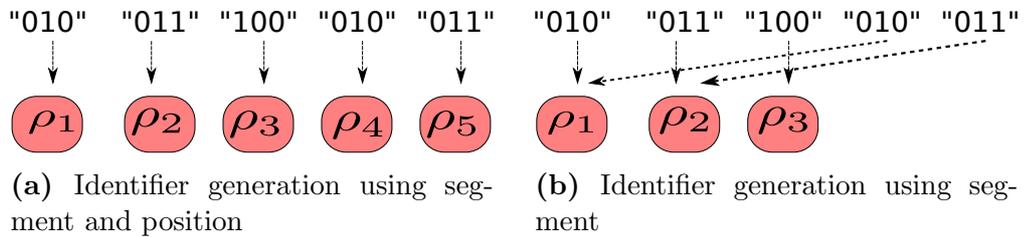


Figure 3.4: An example of identifier generation by using pair of segment and position, or by using only segment as input. In the latter approach, only three bias elements are created instead of five in the former. This is because the segment “010” and “011” have occurred two times in the pattern and a bias element is created when the first time the segment occurs in the pattern in the identifier generation by using only segment.

Identifier ψa and identifier ψb which are both created for the same input \hat{x} are the same, despite the fact that they are created at two different hosts (host a and host b), or at two different times (at time $t = a$ or time $t = b$). As an identifier can be created at any host independently, this scheme provides a better robustness compared to the traditional identifier generation in the HGN and the GN. An identifier of bias array

in the HGN and the GN depends on the number of entries in the bias array^{3.7}. It is created by communicating the identifiers from neighbours within a fixed topology and thus requires strict synchronisation. Any changes to a neighbouring bias array may point to the incorrect entry at other bias arrays.

3.1.6 Network-wide P2P-GN

In the network-wide P2P-GN implementation, a system’s storage space involves all peers where bias elements are distributed among these peers throughout the network. Network-wide associative memories are built by linking such pieces of memory (bias element). The bias elements are arranged in the form of DHT, which manages identifier-value pairs as commonly used for the resource management within structured P2P networks. The DHT is used to systematically index peers and bias elements across a structured P2P network. It improves the network scalability and efficiency in linking the bias elements within a large network.

There are a number of P2P overlay networks which use DHT in their operation such as Chord [179], Pastry [160] or Tapestry [218]. We specifically use Chord overlay in this thesis because of its simplicity and its ability to perform key lookups efficiently. Nonetheless, it is possible to use any other structured P2P overlay network. Despite that there are a number of improvements on DHT-based P2P networks that have been found from the literature which are clearly beneficial to our approach—since our algorithm performance rely on the connections between nodes—we decided to use the original Chord to describe our method here to ease the understanding of our work [50, 175].

In Chord, identifiers are arranged on a circle with size modulo 2^k where k is the size of the identifier. Let \mathcal{I} be the network identifier space; bias elements in space \mathcal{R} are mapped by using a hash function $\text{hash}(\mathcal{R}):\mathcal{R} \rightarrow \mathcal{I}$ to the identifier in space \mathcal{I} . The $\text{hash}(\mathcal{R})$ is defined by the overlay network design. Here, a similar hash

^{3.7}In the HGN and the GN, an identifier for a pattern is created when the pattern is added into the bias array. Given the size of the bias array is i prior to the insertion, the identifier ψ of a pattern \hat{x} is $i + 1$.

function that is used in the network identifier generation is used to generate the bias identifier (i.e., $\mathbf{q}(\cdot)=\text{hash}(\cdot)$). Therefore, the bias identifier space equals to the network identifier space (i.e., $\mathcal{R} = \mathcal{I}$) and the mapping $\mathcal{R} \rightarrow \mathcal{I}$ is unnecessary. A bias element is assigned to the first peer whose identifier follows the bias element's identifier in clockwise.

Each peer has a finger table with k entries that maintains a successor list. Let $p_{suc[i]}$ be the i th successor of a peer p where $p_{suc[0]}$ is the closest successor of p . \mathcal{I} identifier space is partitioned into subspaces where a peer p with identifier $p.ID$ manages subspace $[p.ID, p_{suc[0]}.ID)$. Thus, a bias element r with identifier $r.ID$ and $p.ID \leq r.ID < p_{suc[0]}.ID$ are located at p .

In this scheme, every peer becomes the root of its own queries and thus the computation is fully-distributed. As a result, this improves the fault-resilience and efficiency of the algorithm. This also led to an asynchronous computation which complies with the asynchronous P2P network computing nature. Our approach also improves the system scalability by distributing the computational load across the network. The use of DHT facilitates a systematic workload distribution where data is located at predefined locations. Hence the location of each stored memory is deterministic and controlled; this helps to avoid an uncontrolled data redundancy. The DHT-based data distribution effectively improves the algorithm scalability by averaging the local overhead. Hence, the workload per peer is decreasing with the growing of the network size and vice-versa.

3.1.7 Learning Procedure

The learning process is explained in Algorithm 3.1 and Algorithm 3.2. The input to the learning algorithm are a set of sub-patterns $\{\hat{x}_i\}_{i=1}^{n_H}$ and their class label c . Note that these sub-patterns have a similar class label since they are constructed from the same input pattern \mathbf{x} . During learning, a requester generates bias identifier for n_H input sub-patterns $\{\hat{x}_i\}_{i=1}^{n_H}$. Here, we hash the pair of sub-pattern and its position ($(\langle \hat{x}_i, i \rangle)$)—using the hash function as provided by P2P overlay design—to

Algorithm 3.1. P2P-GN Learning

```

1: Input:  $\{\hat{x}_i\}_{i=1}^{n_H}$  (input sub-patterns),  $c$  (class label)
2:
3: At a requester peer  $p$ 
4: for all  $\{\hat{x}_i\}_{i=1}^{n_H}$  do
5:   Generate the bias identifier —  $\psi(i) \leftarrow \{(\mathbf{q}(\hat{x}_i, i).$ 
6:   Send learning request for  $\langle \psi(i), c \rangle$  using LEARN_REQUEST.
7: end for
8:
9: At a leaf node  $b$ : upon receiving LEARN_REQUEST for  $\langle \psi(i), c \rangle$ 
10: if  $b.ID \leq \psi < b_{successor[0].ID}$  then
11:   Execute localLearning(.)
12: else
13:   Forward to another peer by using routing service.
14: end if

```

Algorithm 3.2. localLearning(ψ, c)

```

1: if bias element ( $\psi$ ) is found then
2:   Get the score table( $\psi$ ).
3:   if  $l$  is found in the score table ( $\psi$ ) then
4:     Update the score table  $\rightarrow \hat{f}(\psi, c) = \hat{f}(\psi, c) + 1$ .
5:   else
6:     Create a pair  $\langle c, 1 \rangle$  and add it into score table ( $\psi$ ).
7:   end if
8: else
9:   Create a new bias element  $\langle \psi \rangle$  with a score table consists an entry  $\langle c, 1 \rangle$ .
10: end if

```

get the bias identifier of the sub-pattern \hat{x}_i . The generated identifier is sent together with its labels (in the form of a *bias query*) via LEARN_REQUEST message, which uses the lookup service, i.e., provided by the overlay protocol. The lookup service then discovers the peer which responsible to store the bias element of the queried identifier.

Definition 3.8 (bias query). *A pair $\langle \psi, c \rangle$ that is sent by the LEARN_REQUEST and RECALL_REQUEST messages.*

where ψ is a bias identifier and c is the class label.

Definition 3.9 (LEARN_REQUEST(ψ, c)). *A LEARN_REQUEST(ψ, c) is a lookup message to locate a peer p with identifier $p.ID$ where $p.ID \leq \psi < p_{successor[0].ID}$ and it contains a request for learning of a bias identifier ψ of label c .*

Upon receiving a `LEARN_REQUEST` message for $\langle \psi, c \rangle$, the receiving peer b checks if ψ is within its subspace, i.e., $b.ID \leq \psi < b_{successor[0].ID}$. If it is not, then it forwards the message to another peer by using routing service (provided by the P2P system). Otherwise, the peer b performs a `localLearning(.)` function where it first checks if the bias element for ψ is available in its depository. If the bias element is not found, then a new bias element is created for ψ with a score table consists an entry $\langle c, 1 \rangle$ (where $\hat{f}(\psi, c) = 1$ since this is the first time the class c occurs with identifier ψ). However, if the bias element is found, then it further identifies if its score table has already stored label c . If it has not been stored, then a pair $\langle c, 1 \rangle$ is created and inserted into the store table; otherwise, the count for $\hat{f}(\psi, c)$ is increased.

Figure 3.5 describes the learning procedure in a graphical example. Let $\{\mathbf{x}, c\}$ be a training instance where $\mathbf{x} = \{a, b, d, c, f, g, h\}$ and class $c = \text{“Jazz”}$. Let d_s be 3 and $OV = 2$. Then, the sub-patterns are $\hat{x}_1 = \{a, b, d\}$, $\hat{x}_2 = \{b, d, c\}$, $\hat{x}_3 = \{d, c, f\}$, $\hat{x}_4 = \{c, f, g\}$ and $\hat{x}_5 = \{f, g, h\}$. The generated identifiers for these sub-patterns are $\psi(1) = 46$, $\psi(2) = 16$, $\psi(3) = 20$, $\psi(4) = 22$, and $\psi(5) = 25$, respectively. A requester peer 40 sends the bias queries for these identifiers with the label c to the network by using `LEARN_REQUEST` messages. The peer 15 receives bias query 16; peer 18 receives bias query 20; peer 21 receives bias query 20; peer 24 receives bias query 25; and peer 46 receives bias query 43. After verifying that the received identifier is within its subspace, the peer then executes then `localLearning(.)` where the bias element of the identifier is updated. In case that none of these identifiers have occurred before, then five new bias elements are created by the responsible peers for the subspaces of these bias elements, i.e., peers 15, 18, 21, 24 and 46.

3.1.8 Recall Procedure

The recall procedure consists of three main functions: (i) `localRecall(.)`, (ii) `agg(.)` and (iii) `finalPrediction(.)`. Function `localRecall(.)` produces a local prediction for a received query at a peer using its local classifier. In the function `localRecall(.)`, given a sub-pattern \hat{x} is queried at a peer p , the output is the prediction of the

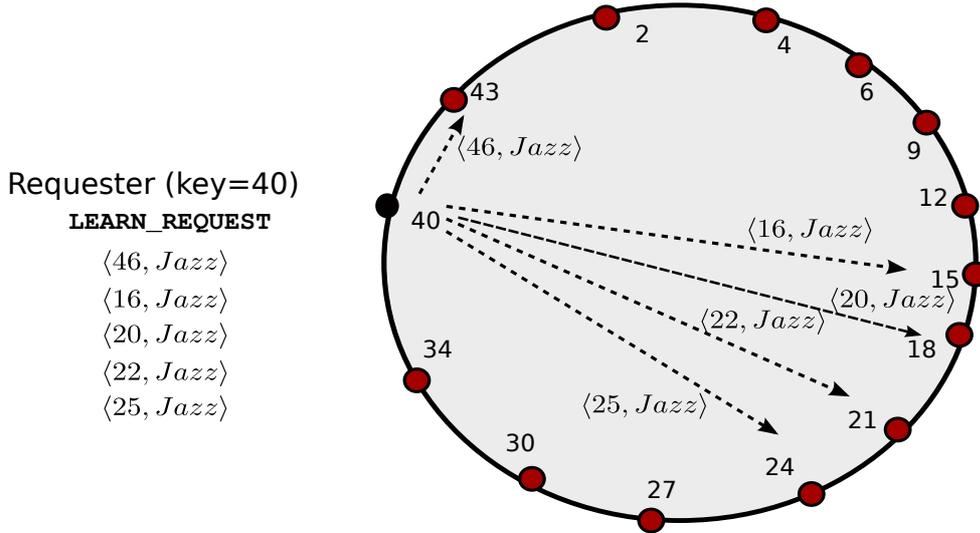


Figure 3.5: An example of learning request from peer 40 for sub-patterns with identifiers $\{46, 16, 20, 22, 25\}$ and label “Jazz”. The `LEARN_REQUEST` messages from peer 40 will be received by peer 15 for bias query 16, peer 18 for bias query 20, peer 21 for bias query 20, peer 24 for bias query 25 and peer 46 for bias query 43. Assume that these bias queries have never been seen before, then these peers create five new bias elements and store the associated label in the score table of each bias element.

class associating with the sub-pattern \hat{x} with its prediction confidence. While in the function `agg(.)`, given predictions from several peers for a sub-pattern \hat{x} , the output is the combined predictions on the class of the queried sub-patterns. Function `finalPrediction(.)` aims to determine the most likely class based on the combined predictions.

The pseudo code of the P2P-GN recall algorithm is given in Algorithm 3.3. After generating identifiers of all n_H segments, a requester peer sends n_H `RECALL_REQUEST` messages—for $\{\psi(i)\}_{i=1}^{n_H}$ where $\psi(i)$ is the identifier for the i th segment—in parallel. The receiving leaf node then verifies if the identifier is within its subspace. If it is not, then the leaf node forwards the query to another peer by using the routing services, else it performs `localRecall(.)` and sends the response with prediction back to the requester. The requester then aggregates the received predictions using `agg(.)` and computes a global prediction using `finalPrediction(.)`.

Algorithm 3.3. P2P-GN Recall

```

1: Input:  $\{\hat{x}_i\}_{i=1}^{n_H}$  (input sub-patterns)
2:
3: At a requester peer  $p$ 
4: for all  $\{\hat{x}_i\}_{i=1}^{n_H}$  do
5:   Generate a bias identifier —  $\psi(i) \leftarrow \mathbf{q}(\hat{x}, i)$ .
6:   Send query for  $\psi(i)$  using RECALL_REQUEST.
7: end for
8:
9: At a leaf node  $b$ : upon receiving RECALL_REQUEST for  $\psi$ 
10: if  $b.ID \leq \psi < b_{successor[0].ID}$  then
11:   if bias element ( $\psi$ ) is found then
12:     Calculate prediction  $\hat{r} = \text{localRecall}(\psi)$ .
13:   else
14:      $\hat{r} = \{\text{"unknown"}, 0, 0\}$ .
15:   end if
16:   Send the RECALL_RESPONSE( $\hat{r}$ ) to the requester peer  $p$ .
17: else
18:   Forward to another peer by using routing service.
19: end if
20:
21: At requester peer  $p$ : upon receiving RECALL_RESPONSE( $\hat{r}$ )
22: Receive predictions  $\{\hat{r}(i)\}_{i=1}^{n_H}$ .
23: Calculate prediction  $\hat{R}(G)$  using  $\text{agg}(\{\hat{r}(i)\}_{i=1}^{n_H})$ 
24:  $\mathcal{L} = \text{finalPrediction}(\hat{R}(G))$ .

```

Definition 3.10 (`RECALL_REQUEST` (ψ)). A `RECALL_REQUEST`(ψ) is a lookup message to locate a peer p with identifier $p.ID$ where $p.ID \leq \psi < p_{successor[0].ID}$ and contains a request for recall of a bias identifier ψ .

Definition 3.11 (`RECALL_RESPONSE` (\hat{r})). A `RECALL_RESPONSE`(\hat{r}) is a message that is sent from a peer p_a to a peer p_b in response to the `RECALL_REQUEST` and contains a local prediction.

Figure 3.6 provides an example of recall requests where a peer 6 wants to get prediction for identifiers $\{46, 16, 20, 22, 25\}$. It sends five recall request messages $\{\text{RECALL_REQUEST}(46), \text{RECALL_REQUEST}(16), \text{RECALL_REQUEST}(20), \text{RECALL_REQUEST}(22), \text{and } \text{RECALL_REQUEST}(25)\}$ to the network. These request messages are received by peer 15, peer 18, peer 21, peer 24, and peer 43, respectively. Then, these peers reply with their predictions to the peer 6. These processes are performed in parallel,

hence a reasonably fast processing time is expected despite the potential delay of lookup processes.

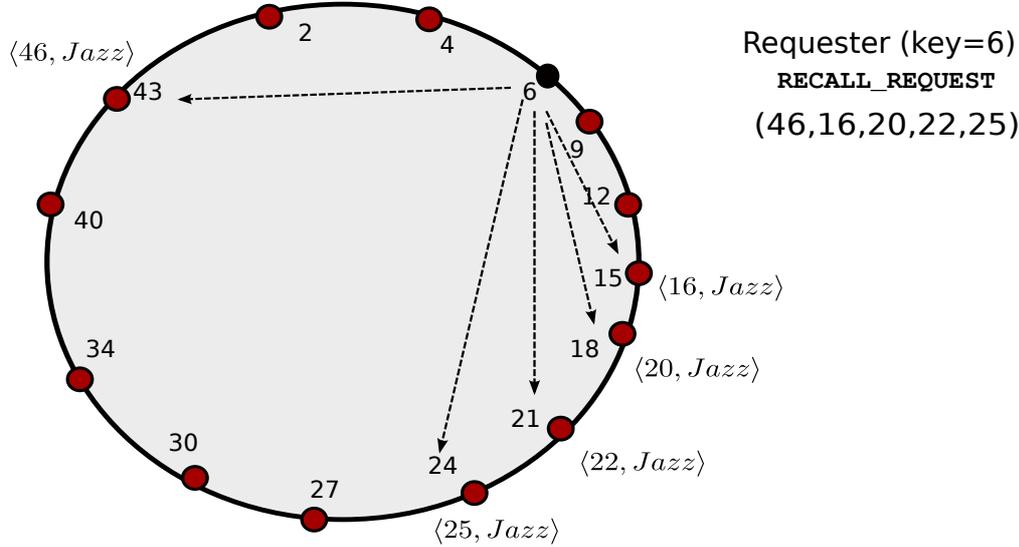


Figure 3.6: An example of recall requests. A peer 6 submits five recall requests for identifiers $\{46, 16, 20, 22, 25\}$; these requests arrive at peer 15, peer 18, peer 21, peer 24, and peer 43.

Local Recall

During local recall, the `localRecall(.)` function (see Algorithm 3.4) produces a prediction element which is recorded in a form of 3-tuple $\langle c, v(c), w(c) \rangle$, where c is the predicted class while $v(c)$ and $w(c)$ is the vote and weight for the class c respectively. A prediction \hat{r} may consists of more than one prediction element and thus, it is a

Algorithm 3.4. `localRecall(ψ)`

- 1: Get a score table(ψ).
 - 2: $i = 0$.
 - 3: **while** $i < |\mathcal{C}|$ **do**
 - 4: Calculate $v(c_i)$.
 - 5: Calculate $w(c_i)$.
 - 6: Add $\langle c_i, v(c_i), w(c_i) \rangle$ to \hat{r}
 - 7: $i++$
 - 8: **end while**
 - 9: Return \hat{r} .
-

set of prediction element $\langle c_i, v(c_i), w(c_i) \rangle \in \hat{r}$ where c_i is a i th class, or in vector representation $\hat{r} = \overrightarrow{\langle c, v, w \rangle}$.

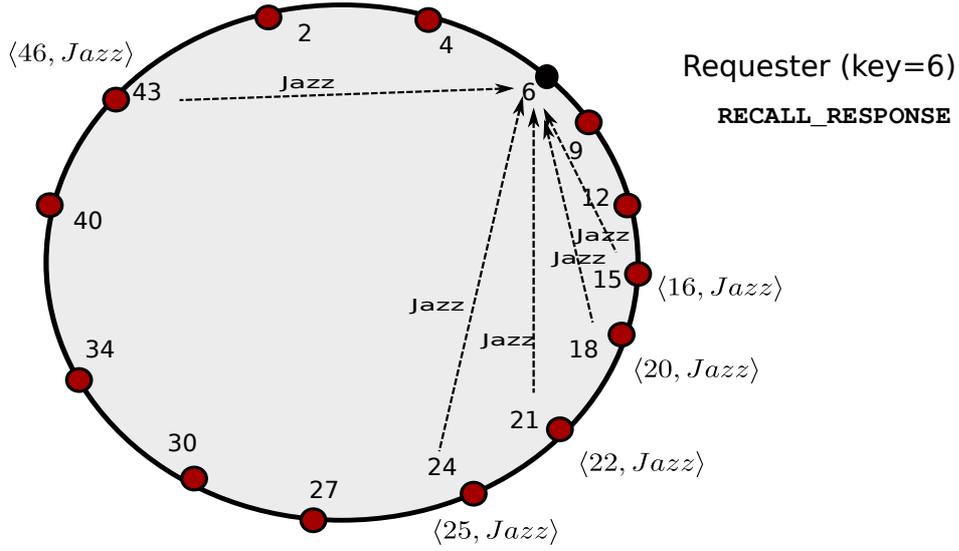


Figure 3.7: An example of responses from recall requests at Figure 3.6. Peer 15, peer 18, peer 21, peer 24, and peer 40 send the prediction “Jazz” directly to the peer 6.

Given a bias query for an element ψ , the local recall is performed at a leaf node. When ψ is found, then the corresponding score $\text{table}(\psi)$ is used to calculate the output prediction. It is possible to have several predicted classes with the same vote and highest weight in an output prediction \hat{r} . Given \mathcal{C} is a set of classes and $c \in \mathcal{C}$, $v(c)$ (see line(4) of Algorithm 3.4) is

$$v(c) = \begin{cases} 1 & \text{if } \psi \text{ is found in the score table}(\psi) \text{ and } c \in \mathcal{C} \\ 0 & \text{otherwise} \end{cases}$$

Then, given $\hat{f}(\psi, c)$ is the frequency of the sub-pattern with identifier ψ has occurred with class c , and c_i is the i th class in the set \mathcal{C} , $w(c)$ (see line(5) of Algorithm 3.4) is calculated as follows

$$w(c) = \frac{\hat{f}(\psi, c)}{\sum_{i=1}^{|\mathcal{C}|} \hat{f}(\psi, c_i)}$$

Aggregation

Let $\{\hat{r}(1), \hat{r}(2), \dots, \hat{r}(n_H)\}$ be a set of local predictions where $\hat{r}(i)$ is a local prediction from node i ; $\hat{R}(G)$ be the aggregated local predictions $\hat{R}(G) := \mathbf{agg}(\hat{r}(i)_{i=1}^{n_H})$ which are combined using *aggregate rule*. Distributed versions of some complex algorithms like SVM and decision trees usually require a sophisticated aggregation technique [66, 17, 8]. In contrast, the scheme in this thesis is simple, and thus the adopted aggregation rule is straightforward and simple as defined in the aggregate rule as following.

Aggregate rule for function $\mathbf{agg}(\hat{r}(i)_{i=1}^{n_H})$ Using *sum* rule, the aggregated vote $agg_v(c)$ and aggregated weight $agg_w(c)$ for class c are computed as follows:

$$agg_v(c) = \sum_{i=1}^{n_H} v(c)_i$$

where $v(c)_i$ is the vote for class c that is received from node i and n_H is the number of inputs.

$$agg_w(c) = \frac{\sum_{i=1}^m w(c)_i}{\sum_{j=1}^{|C|} \sum_{i=1}^{n_H} w(j)_i}$$

where $w(j)_i$ is the weight for class j that is received from node i and n_H is the number of inputs.

Final Prediction

The final prediction is made using the function `finalPrediction(.)` after aggregation at the root node or requester node based on two stage predictions (see Figure 3.8): (i) majority voting—selecting the prediction with the majority vote, \hat{R}_{max} ; and (ii) selecting an option from \hat{R}_{max} with the highest-weighted sum. The former is applied first through the `selectMaxVote(.)` function to obtain a set of predictions, \hat{R}_{max} with the highest vote and the latter is used whenever there are more than one options in \hat{R}_{max} . Then, the final prediction \mathcal{L} is calculated using function `selectMaxWeight(.)`.

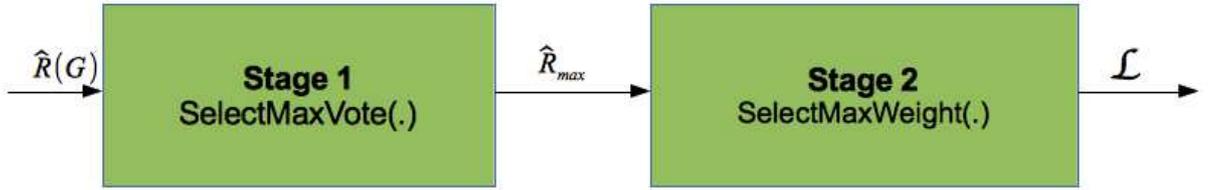


Figure 3.8: The two stage predictions during the final prediction process. The first stage performs `selectMaxVote(.)` function and the second stage performs `selectMaxWeight(.)`. The input to the first stage is $\hat{R}(G)$; the first stage produces \hat{R}_{max} which then becomes an input to the second stage. The second stage produces \mathcal{L} that is the final prediction.

selectMaxVote(.) function. $\hat{R}_{max} := \text{Select } \{c, \text{agg-}v(c), \text{agg-}w(c)\} \in \hat{R}(G)$ with $\text{agg-}v(c) \geq \text{max}_v - e$ where a prediction $\{c, \text{agg-}v(c), \text{agg-}w(c)\} \in \hat{R}(G)$, max is the highest $\{\text{agg-}v(c)\}_{c \in \mathcal{C}}$, e is a discount integer value and $0 \leq e < \text{max}_v$.

selectMaxWeight(.) function. $\mathcal{L} := \text{Select } \{c, \text{agg-}v(c), \text{agg-}w(c)\} \in \hat{R}_{max}$ with $\text{agg-}w(c) = \text{max}_w$ where max_w is the highest $\{\text{agg-}w(c)\}_{c \in \mathcal{C}}$.

3.2 Experimental Results

A series of experiments were conducted to evaluate our scheme against four main evaluation criteria: accuracy, efficiency (communication and run-time), invariance to imbalanced data distribution, and adaptivity to various network scale. First, we compare the accuracy of the P2P-GN with several centralised classifiers (ID3, k -NN, naive Bayes, 1-NN, RBFN and BPNN). This also demonstrates its ability to work effectively with in the absence of a server. Second, we show that our method is efficient with respect to the communication overheads and recall (classification) time compared to the Ivote-DPV and semi-distributed ensemble algorithms. Third, we evaluate the performance of our algorithm within imbalanced data distribution and we compare it with distributed algorithms—Ivote-DPV and three ensemble algorithms (ensemble k -NN, ensemble naive Bayes and ensemble ID3). The results show that our method is highly invariant to imbalanced data distribution compared

to these distributed algorithms. Finally, we demonstrate the advantage of using DHT and fine-granularity distribution of components in this work; which allows for the adaptivity of the algorithm for various scale P2P networks. This contrast with the cluster-based implementation with fixed structure, which is inherent in the GN based algorithms.

3.2.1 Simulator and Classifiers Setting

All experiments were run on a PC with Intel Core i-7 2.9Hz and 8Gb memory and they were implemented using Java. The distributed algorithms were deployed on the Peersim simulator [136] to simulate the P2P environment. We implemented two versions of the P2P-GN implementations: *P2P-GN with Chord* and *P2P-GN with optimal connection*.

P2P-GN with Chord The P2P-GN with Chord uses the Chord’s lookup function to link the bias elements, therefore, the number of messages includes the number of hops for lookups and the run-time delay includes the delay for lookups.

P2P-GN with optimal connection In the P2P-GN with optimal connection, the location of the bias elements are assumed to be known by a requester. Hence, direct connections within a single hop can be established.

Unless mentioned otherwise, the default settings of the P2P-GN simulation in this experiment is the P2P-GN with Chord network with the following network parameters: network size equals to 5,000 peers, size identifier equals to 160 bits, and successor list size equals to 12.

We compare our algorithm against six centralised state-of-the-art algorithms: ID3, k -NN, naive Bayes, 1-NN, RBFN, and BPNN. In addition, we also compare the efficiency and scalability of our algorithm for large networks against two distributed algorithms: the Ivote-DPV and ensemble ID3. Either the aggregation is

performed during learning or during recall, most of the available distributed algorithms aim to include all observations as much as possible in their global model or predictions. Here, the distributed algorithms perform aggregation during prediction^{3.8} in all experiments. This is consistent with our aim in Section 1.3 of Chapter 1—to focus on the aggregation during prediction instead of learning phase.

In the ensemble ID3, every peer sends its local solution (classified by ID3 classifiers) to a single site where a single global solution is calculated using majority voting. Although it is semi-distributed and clearly not scalable for large networks, we neglect this issue since it is simple and includes all of the available observations at every peer. This gives us an exact classification that provides a good benchmark on the accuracy of our algorithm. In the experiment in Section 3.2.5, two other ensemble classifiers are used: ensemble k -NN and ensemble naive Bayes.

The Weka [75], an open source software for data mining, was used in all experiments for centralised algorithms and base classifiers. For the centralised BPNN, we set the number of hidden nodes in the BPNN to $(d + |\mathcal{C}|)/2$ where d is the number of attributes and $|\mathcal{C}|$ is the number of classes. All of the distributed algorithms were implemented using Java and were simulated on the Peersim simulator. For the experiments involving ensemble classifiers, the base classifiers were developed using the Weka package and we applied majority voting for aggregation.

For the experiments involving Ivote (except in the experiment for imbalanced dataset in Section 3.2.5), we ensured that the size per bite^{3.9} was not too small since the smaller size per bite results in a very high number of iterations in our preliminary experiments. This is due to the difficulty in making classification on small datasets (insufficient training samples) by the base classifier. Therefore, if the dataset size is larger or equal to 100, we set the size per bite to 20% of the dataset size. Otherwise, the size per bite equals to the size of the local dataset. In the Ivote-DPV experiments, parameter λ was set to 0.002, and the J48 classifier was

^{3.8}The aggregation is performed during the prediction phase when the local prediction results from all sites are combined into a global prediction result.

^{3.9}Size per bite refers to the sample size for each iteration in Ivote.

used for base classifiers, as applied in Luo et al. [125]. The maximum number of learning iterations for Ivote classifier (local classifier) was set to 1,000 iterations, while, the number of neighbours for the DPV was set to 13 which is equal to the size of the successor list (12) plus one predecessor peer.

3.2.2 Datasets

The datasets used for our evaluation are available publicly and they are mainly from the standard machine learning dataset repository UCI database [13]. The selected datasets are among the largest datasets in the UCI database and they were chosen based on the size of the datasets and their high-dimensional (large number of attributes) aspect, which are important to evaluate the scalability of our approach. Additionally, the experiments on these datasets evaluate the effectiveness of our approach for speech recognition problems (on ISOLET (Isolated Letter Speech Recognition)) and handwritten recognition problems (on MFeat (Multiple Features) and MNIST (Mixed National Institute of Standards and Technology) database). The results demonstrate the potential of our algorithm to provide a friendlier human-interface for P2P-based applications. The evaluation on a cancer detection problem (on ARCENE) demonstrates the potential of P2P-based pattern recognition to contribute to health care sector by providing such services, economically—the scalable pattern recognition algorithm is practical considering the large scale of health data which is distributed across the world.

We did not perform any pre-processing of these datasets, except for transforming the continuous values into discrete values, as our scheme only accepts discrete values. A brief explanation on these datasets is given. Table 3.3 then summarised the dimension and number of instances of each dataset.

MNIST. This handwritten digits dataset [109] is one of the largest datasets for pattern recognition in the UCI database, with 70,000 instances and the total number of features is 784. This dataset is divided into two parts: the first part consists of

60,000 examples which were used as the training examples, and the second part consists of 10,000 examples which were used as the test examples. Given ten digits, the solution for this problem aims to identify the correct written digit from many different sets of handwriting. Approximately 250 writers were involved in creating 60,000 training examples. The 10,000 test examples are collected from 250 writers which were disjoint from the writers involved in creating the training examples. The examples are originally given in the image format using centred images of 28×28 pixels. Prior using the dataset in this experiment, we performed discretisation on the pre-processed dataset that were represented in ARFF format [75].

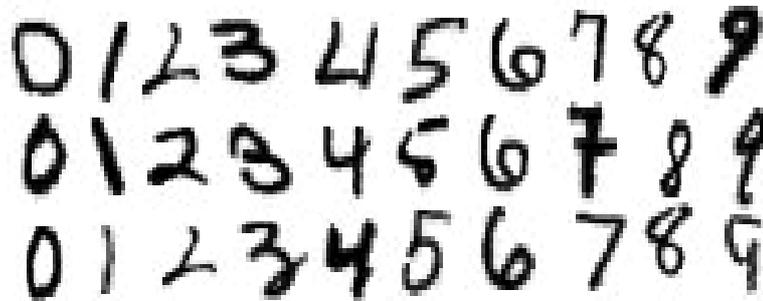


Figure 3.9: Some examples of handwritten images from the MNIST dataset.

In this experiment we chose parameters values based on the maximum value in the Rule 3.3 of Section 3.1.2 where $d_s = \log_2 d$. Hence, we set $d_s = 9$ and we set $OV = 5$ (as $\lceil \frac{9}{2} \rceil$). The resulting P2P-GN structure has 195 leaf nodes.

MFeat. This dataset consists of features of handwritten numerals ('0'–'9'), which were extracted from binary image collections of Dutch utility maps. These datasets are represented by the following six sub-feature sets:

- (a) 240 pixel averages in 2×3 windows.
- (b) 76 Fourier coefficients of the character shapes.
- (c) 47 Zernike moments.

(d) 6 morphological features.

(e) 216 profile correlations.

Due to the large number of features ($d = 2,000$ attributes in total), we selected the parameter values based on the maximum value in the Rule 3.3 of Section 3.1.2; d_s is $\log_2 d = 10$ and OV is 5. The structure in this task consists of 143 leaf nodes.

ARCENE. The task of this dataset is to detect cancer from mass-spectrometric data. It has a very large number of attributes (10,000 attributes) and this makes it a suitable dataset for high-dimensional evaluation. As only training and validation are provided with the label, we only include the training set and validation set in this evaluation. We combined both datasets into a single dataset which consists of 900 examples.

Given a very large dimension, the chosen parameter for this task is based on the maximum value in the Rule 3.3 of Section 3.1.2 which gives $d_s = 13$ and $OV = 8$ to ensure the computational cost per peer remains small. The resulting structure consists of 1,999 leaf nodes.

ISOLET. This high-dimensional dataset for spoken letter speech recognition called ISOLET [64] consists of 617 features and 26 classes. This makes it an ideal dataset to test the scalability of the algorithm. The set of features include spectral coefficients; contour features, sonorant features, pre-sonorant features and post-sonorant features. The experiments on ISOLET dataset was performed using 6,328 for training and another 1,559 for testing as it was originally used for testing in Fanty and Cole [64].

As ISOLET also consists of a large number of attributes (617 attributes), we chose $d_s = \log_2 d = 9$ and $OV = 5$ as given by the maximum value in Rule 3.3 of Section 3.1.2. The structure in this task consists of 153 leaf nodes.

In addition to these datasets, we also used a data generator, the Waveform data generator (Version 2) which is also available publicly in UCI database. By

Table 3.3: Datasets for accuracy testing.

Dataset	Data Dimension	Number of Instances	Number of Classes
ISOLET	617	7,797	26
ARCENE	10,000	200	2
MFeat	2,000	586	10
MNIST	784	70,000	10

using this data generator, we are able to generate a large number of samples to simulate a distributed dataset across a large network. This dataset was used in the experiments involving scalability evaluation for large networks (communication efficiency and time efficiency).

Waveform generator (Version 2) The Waveform database generator was originally presented in [25]. However, we used the version 2 of the waveform generator which includes 19 noise attributes in addition to the original 21 waveform attributes (from the Waveform generator (Version 1)). Hence, the resulting dataset consists of 40 attributes and three classes where the class distribution of this dataset is 33% for each class. Here, we generated 660,000 training samples and 5,000 testing samples. The P2P-GN structure for this problem consists of 38 leaf nodes since we selected parameters: $d_s = 3$ and $OV = 2$.

3.2.3 Comparative Accuracy Against Centralised Algorithms

The accuracy test is conducted to show that the proposed distributed scheme is comparable against other state-of-art algorithms. As these algorithms are centralised where the whole dataset at a single location, they clearly have more advantage compared to our distributed scheme. However, our intention is not to compete with these centralised algorithms in terms of accuracy, but to show that the accuracy of our scheme is comparable against these well-known algorithms despite operating within a fully-distributed approach. The accuracy metric in this experiment is

obtained as follows.

$$\text{accuracy} = \frac{\text{number of correctly predicted items}}{\text{total number of tests}} \quad (3.2)$$

The experiments for the Arcene, ISOLET and MFeat datasets were performed using 10-fold cross validation and we used the cross-validation package from the Weka to split our test and training sets in our distributed simulations. The use of the Weka package to split the dataset ensures that exactly same set of evaluations are used in experiments for every algorithm. For the MNIST, we used the hold-out evaluation method. The large number of examples in these datasets allows for a correct evaluation despite the use of hold-out method instead of 10-fold cross evaluation. The original train and test datasets were used from the MNIST dataset.

However, we were unable to process the high-dimensional datasets including the Arcene, MNIST, MFeat and ISOLET due to the high computational complexity of BPNN and RBFN. Therefore, for the experiments involving BPNN and RBFN, these datasets underwent a feature selection process using correlation-based feature selection [76]. This feature selection tool is provided by Weka. The experiments for BPNN for these datasets were conducted by using only ten learning iterations for the same reason.

Table 3.4 compares the accuracy of the P2P-GN against six centralised, state-of-the-art classifiers on five different datasets. As can be seen, the P2P-GN performs

Classifiers	Dataset			
	ISOLET	Arcene	MFeat	MNIST
BPNN	0.95	0.88	0.98	0.71
RBFN	0.92	0.86	0.95	0.84
1-NN	0.81	0.89	0.97	0.90
k -NN	0.84	0.88	0.90	0.91
ID3	0.66	0.92	0.84	0.64
Naive Bayes	0.90	0.70	0.94	0.84
P2P-GN	0.82	0.85	0.96	0.89

Table 3.4: Accuracy comparison with centralised, state-of-the-art classifiers

better than 1-NN and ID3 on the ISOLET dataset and it performs better than RBFN, k -NN, naive Bayes and ID3 on the MFeat dataset. Our algorithm is also more accurate than BPNN, RBFN, naive Bayes and ID3 on the MNIST dataset.

The fact that the whole dataset is located at a single host in centralised state-of-the-art algorithms gives an advantage to these algorithms, in comparison to our scheme (where datasets are distributed at several different locations). However, the results show the comparable accuracy of our distributed scheme against the centralised state-of-the-art algorithms which fulfils our objective in this experiment.

3.2.4 Evaluating Efficiency for Large Networks

This section presents the results to evaluate our algorithm with respect to the large network scalability by investigating its communication efficiency and time efficiency. Here, we compare our method with an ensemble ID3 and the Ivote-DPV. In the ensemble algorithm, all peers send their local prediction to a single peer where a single global prediction is made through majority voting.

The dataset was generated using the Waveform data generator (Version 2) to generate a dataset with a large number of samples—we generated 660,000 samples for training. First, we randomly assigned peers’ local dataset size (number of samples for every peer) by using a uniform distribution with upper bound of 500 samples per peer. Second, for each peer, we selected l samples using sampling by replacement from the generated training samples (660,000 samples) where l is the peer’s local dataset size (as determined in the first step).

We measure the algorithm’s communication efficiency for large-scale distributed system by using two metrics: \tilde{m} , that is the average number of messages per query and \tilde{u} , that is the average communication load^{3.10} per query by increasing the network size from 500 to 3,500. We measure these metrics during recall since the communication overhead during recall is known to be larger than during learning. The number of messages during learning involves n_H ^{3.11} LEARN_REQUEST messages,

^{3.10}The size of transferred data in a message.

^{3.11} n_H is the number of leaf nodes or segments.

while, the number of messages during recall involves n_H `RECALL_REQUEST` messages and n_H `RECALL_RESPONSE` messages. It is also known that the communication load per learning is constant (since it only consists of a bias identifier and a class label), while, the communication load per recall can be vary (since it consists of a number of possible classes with their votes and weights in the local predictions). Hence, the communication overhead during recall is worth investigated compared to during learning.

The metrics \tilde{m} and \tilde{u} are calculated as follows.

$$\tilde{m} = \frac{M_{recall}}{n_{recall}} \quad (3.3)$$

where M_{recall} is the total number of messages during recall (includes the lookup costs) and n_{recall} is the total number of recall tests^{3.12} across the network during the simulation time.

$$\tilde{u} = \frac{\tilde{l}_{recall}}{n_{recall}} \quad (3.4)$$

where \tilde{l}_{recall} is the total communication load during recall. Every unit of the communication load represents a class entry in a message which contains a prediction. Thus, every message in a local prediction message contains k units where in the Ivote-DPV, $k = d$, while, in the P2P-GN, $0 \leq k \leq d$.

The recall was started after the learning completed. A recall test was sent for every 50 seconds from random peers. In total, 5,000 tests are submitted by the end of the experiment. From the results in Figure 3.10 and Figure 3.11, we found that the P2P-GN algorithm uses less messages and less communication load per query, than the ensemble k -NN and the Ivote-DPV. The communication load per query is low in the P2P-GN since the messages containing predictions are sent directly to the requester^{3.13} whereas the predictions are exchanged among peers in the Ivote-DPV in several iterations. It can be summarised that the average number of messages and

^{3.12}One recall test for a single test instance.

^{3.13}The lookup messages are only used to send the queries. The receiving peers then establish direct connections to the requester peer to send their predictions.

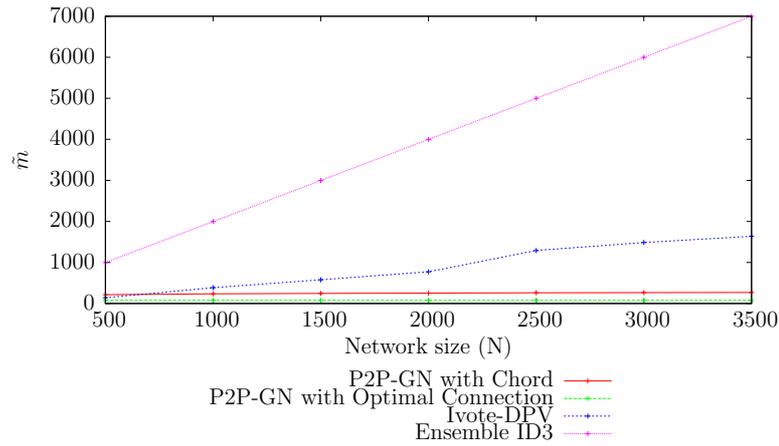


Figure 3.10: \hat{m} in the P2P-GN, Ivote-DPV and ensemble ID3 during recall on the dataset generated by the Waveform data generator (Version 2).

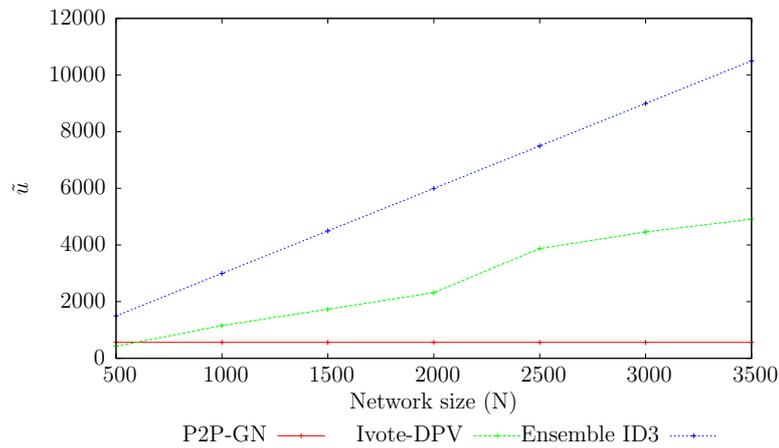


Figure 3.11: \hat{u} in the P2P-GN, Ivote-DPV and ensemble ID3 during recall on the dataset generated by the Waveform data generator (Version 2).

communication load in the P2P-GN algorithm grow very slowly with an increase in the network size. The ensemble ID3 incurs the highest number of messages and communication load. This is because a requester peer sends its request for recall to all peers and then every peer responds with its local prediction to the requester. However, in the Ivote-DPV, the average number of messages and communication load grow slower than the ensemble ID3.

Figure 3.12 provides the comparison on the performance of the P2P-GN with optimal connection and the P2P-GN with the Chord overlay. Currently, the optimal

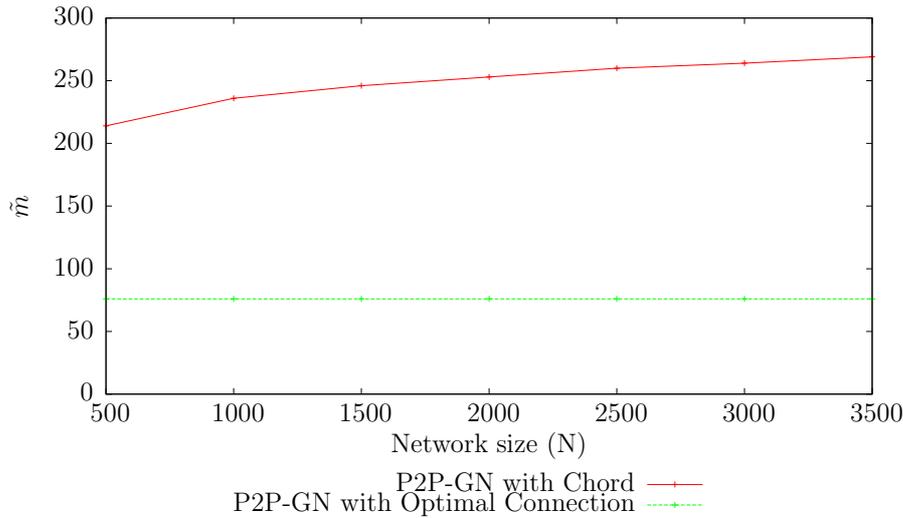


Figure 3.12: \tilde{m} in the P2P-GN with Chord and P2P-GN with optimal connection during recall on the dataset generated by the Waveform data generator (Version 2).

connection implementation, within large P2P networks is impractical and unrealistic since it requires an expensive overheads to allow every peer to know all other peers. However, this assumption provides a benchmark of the best performance of the P2P-GN can achieve with an improvement in the P2P network’s lookup performance. This result shows an improvement in the P2P overlay lookup performance directly improves the communication overhead and vice versa. The communication delay for lookup in this work is, however, an underlying P2P overlay performance matter and in the DHT-based network, it generally increases logarithmically in an increase in network size. We further explore the scalability of our algorithm for large networks with respect to its time efficiency and compare it with the Ivote-DPV and the ensemble ID3. The evaluation metric that we use here is the average time taken for recall per instance, \tilde{t}_{recall} . In this experiment, the communication delay per hop is assumed to be uniformly distributed within 1 to 20 milliseconds. Every peer is assumed can only make at most 100 simultaneous connections. By varying the network size N —by gradually increased N from 500 to 3,500 by 500—we obtain the growth in recall time per instance with an increase in network size. The results are reported in Figure 3.13.

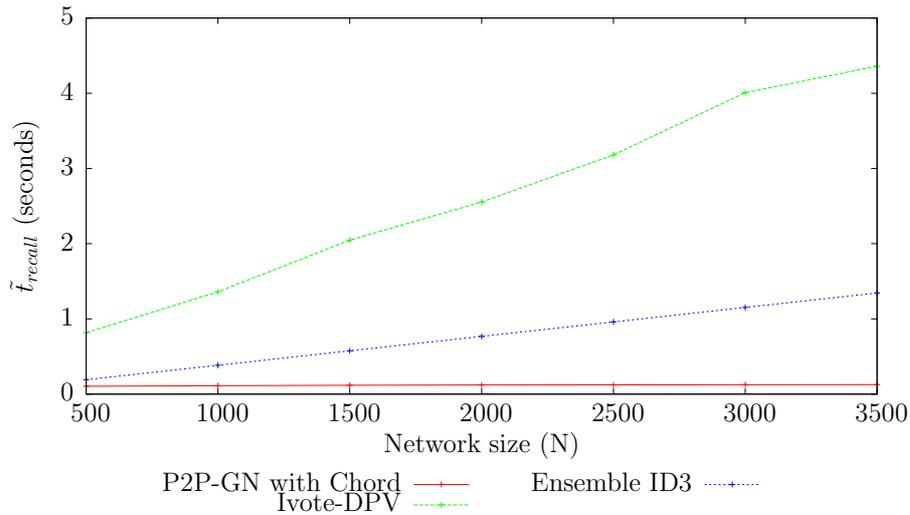


Figure 3.13: Average recall time per instance (\tilde{t}_{recall}) in the P2P-GN with Chord, Ivote-DPV and ensemble ID3 on the dataset generated by the Waveform data generator (Version 2).

Unlike the communication overheads, the recall time in the ensemble ID3 grows slower than the Ivote-DPV. This is because in the ensemble ID3, a requester peer sends at most 100 local recall requests in parallel (since we assumed a peer can make at most 100 simultaneous connections). For the network of 1,000 peers, 10 iterations is required to collect the local predictions from all peers where 100 local predictions are collected per iteration. On the other hand, the Ivote-DPV requires N iterations at the worst case and the high number of iterations results in a slow recall time. Despite that, in practice, the Ivote-DPV is more scalable than the ensemble ID3 since it operates in the absence of a server. Like the Ivote-DPV, the P2P-GN also operates in fully-distributed, but it performs the fastest recall among these distributed algorithms. This is because a requester in the P2P-GN sends 38 simultaneous recall requests into the system, and then the receiving peers process and response to these requests in parallel.

Figure 3.14 shows the slow growth in recall time of the P2P-GN with Chord. The average recall time increases logarithmically with an increase in network size since the recall delay is dominated by the Chord’s lookup delay. In the case of the

P2P-GN with optimal connections, a constant growth in recall time with an increase in network size is expected.

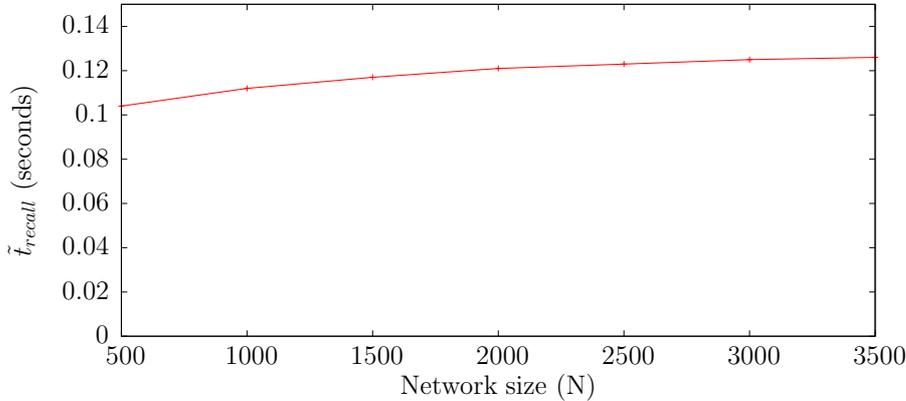


Figure 3.14: Average recall time per instance (\tilde{t}_{recall}) in the P2P-GN with Chord on the dataset generated by the Waveform data generator (Version 2)

3.2.5 Evaluating The Effect of Imbalanced Data Distribution

The experiments in this section aim to show that our approach is invariant to the imbalanced data distribution. In this thesis, we consider two data distribution scenarios that represent the imbalanced data distribution. The data distribution scenarios are described as follows.

- **Scenario 1: Imbalanced class distribution** - not all peers have all classes in their dataset. For instance, some peers' local dataset may consist of instances of class A and class B , while others may have class C and class B .
- **Scenario 2: Small local dataset size** - the data were randomly distributed in uniform to all peers with a very small number of samples per peer.

We generated 60,000 instances for training and 2,000 for testing using the Waveform data generator (Version 2). Since these datasets are large, then we used hold-out method as it is sufficient to prove the accuracy of our method here. The accuracy

metric in this experiment is calculated using Equation 3.2 of Section 3.2.3. The results in this experiment are presented in follows.

Evaluating invariant to the imbalanced class distribution

The aim of this experiment is to measure the accuracy of the distributed classification algorithms when local sites do not have the samples representing all classes. The dataset distribution was set up through the following steps. Initially, we randomly assigned one or two classes for each peer. Then, for every peer, we sampled uniformly at random by replacement from the global training instances with upper bound of 50 samples—the selected samples were ensured within the set of the classes which had been assigned previously to the peer.

Table 3.5: Accuracy test on the imbalanced class distribution scenario.

	Uniform Class Distribution	Imbalanced Class Distribution
P2P-GN	0.824	0.823
Ivote-DPV	0.426	0.387
Ensemble k -NN	0.818	0.326
Ensemble ID3	0.333	0.326
Ensemble naive Bayes	0.775	0.480

We compare the accuracy result of the algorithms which operating on the imbalanced class distribution scenario with the uniform class distribution scenario (where every peer has samples from all three classes and these classes are uniformly distributed). The result in Table 3.5 shows that only the P2P-GN exhibits invariances to the imbalanced class distribution with its accuracy rate decreases by only 0.001. The ensemble k -NN and ensemble naive Bayes exhibit a dramatic loss of approximately 38% to 49% from the accuracy of the uniform class distribution. On the other hand, the accuracy of the Ivote-DPV and ensemble ID3 do not change much, since the rates are low even in the uniform class scenario.

Evaluating invariant to the small local dataset size

In this experiment, every peer was assumed to have data from all of the three classes and the class distribution was uniformly distributed. We varied the bound of local dataset size that gives the maximum number of samples per peer to see the effect of decreasing sample size to the accuracy. The local dataset size bound was set to $\{300, 250, 200, 150, 100, 50\}$ and samples were selected from the global training dataset by using uniform distribution—this also gives uniform class distribution per peer.

Table 3.6: Accuracy test on decreasing sample sizes per peer (with upper bound of 300 to 50 samples per peer).

Classifiers	Local Dataset Size Bound					
	300	250	200	150	100	50
P2P-GN	0.827	0.825	0.827	0.827	0.824	0.824
Ivote-DPV	0.614	0.590	0.556	0.511	0.4512	0.426
Ensemble k -NN	0.824	0.825	0.826	0.821	0.823	0.818
Ensemble ID3	0.460	0.423	0.384	0.362	0.344	0.326
Ensemble naive Bayes	0.804	0.802	0.799	0.788	0.796	0.775

From Table 3.6, we see that despite the poor performance of the Ivote-DPV and the ensemble ID3, their accuracy further decrease with the decrease of the number of samples per peer. The ensemble k -NN and the ensemble naive Bayes show a small reduction of the accuracy when the local dataset size is smaller, but the P2P-GN demonstrates an almost equal accuracy within the range of $0.827 - 0.824$ in all the experiments with different local dataset size. This shows that the P2P-GN performance is not affected by a small local dataset size, compared to other methods.

The results from these experiments suggest the invariant of the P2P-GN approach and the weakness of the Ivote-DPV and the ensemble approaches in dealing with the imbalanced data distribution. The three ensemble classifiers used here also have advantage compared to our approach since they assume the presence of a server-like host. These classifiers combine the results from all peers at the single site—which is communication inefficient and not scalable for large distributed

networks. Hence, the P2P-GN provides the best accuracy and efficient for the imbalanced data distribution scenario within large-scale distributed networks.

3.2.6 Workload Distribution VS Network Size

This experiment investigates the effect of workload distribution with increasing and decreasing network size. In the centralised implementation, all the workload is borne by a single server, while in the semi-distributed system, these workload are partially distributed to all peers and most tasks are done at a single server. In the cluster-based distributed implementation, the workload is distributed among peers in the cluster, but the cluster is burden with a high magnitude of workload when the network is growing. Hence, this experiment aims to show that in our network-wide approach, a large network provides better load balancing of the incurred resources in contrast to the cluster-based, semi-distributed and centralised implementation.

The ISOLET dataset was used in this experiment and for comparison, a cluster-based implementation of the P2P-GN is used—we selected 153 peers (represent leaf nodes as described in Section 4.3) and mapped the leaf nodes to the peers using one-to-one mapping. This is analogous to the bias array structure in GN and an entry in the bias array is a bias element in this thesis (see Khan [99]). The network size was increased from 250, 500, 1,000, 2,000, 4,000 and 8,000. A unit data represents a bias element this experiment. Figure 3.15 shows the portion of network that is used for the cluster-based P2P-GN operation.

As shown in Figure 3.15, about 27.6% peers do not store any bias entry, while, about 11.2% peers stores above 700 entries when the size of the network is 250. When the network size doubles gradually to 500, 1,000, 2,000, 4,000, and 8,000, the percentage of peers with zero bias entry increases to 63.8%, 81.9%, 91%, 95.5% and 97.7% respectively. The workload is only distributed among these peers and although the network size is increasing, the workload at the peers is not distributed to other peers (outside the cluster). This results in the workload to be only distributed among 2.3% from 8,000 peers. Out of this 2.3%, about 17.39% store above 7,000

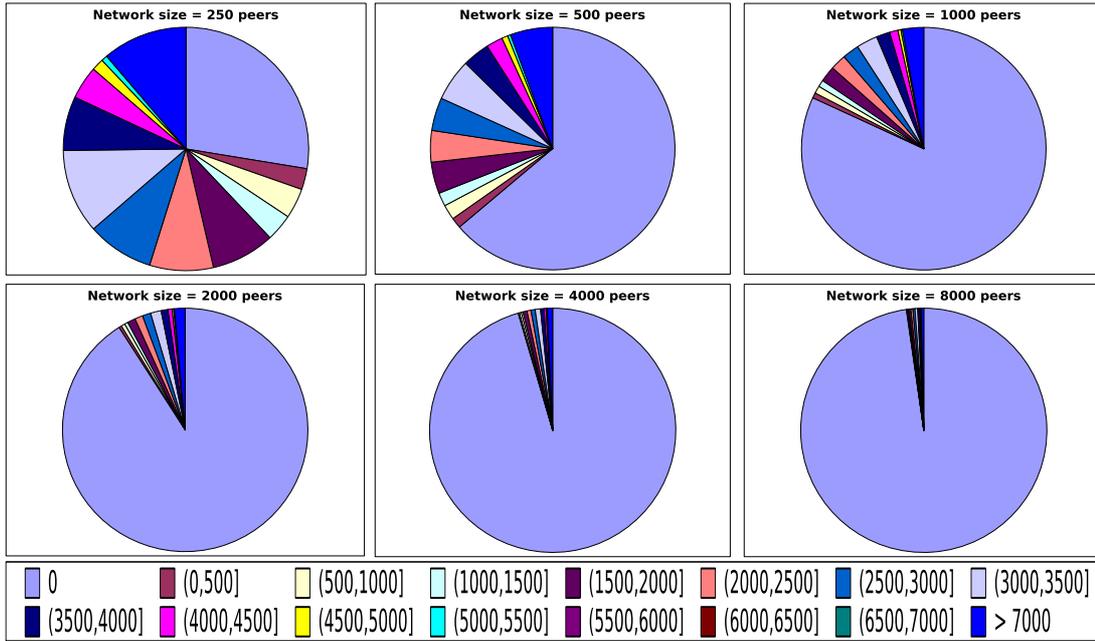


Figure 3.15: Workload (bias elements) distribution throughout the network with an increase in network size in the cluster-based implementation. The load is concentrated among a small subset of peers across the network.

bias entries. As a result, all of the queries concentrate on these cluster and this may burden the cluster; lead to a bottleneck problem when the network becomes very large.

In contrast, we found that the increase in network size reduces the workload per node as shown in Figure 3.16. The percentage of peers which store above 7,000 elements reduces from 5.6% when the network size is 250, 0.2% when the network size is doubled to 500 and zero percentage when the network size is at least 1,000. While the percentage of peers which do not store any bias elements is approximately only 0.2% when size of network is 1,000 and 1.3% when the size of network is 8,000. The number of peers which hold between 1 to 500 entries is increasing from 17.8% when the size of network is 250, 33.2% when the network size is 500, 54% when the network size is 1,000, 80.7% when the network size is 2,000, 95.8% when the network size is 4,000 and finally 98.8% when the network size is 8,000. This shows that our approach is suitable for large-scale P2P environments. However, an extra load balancing effort may be required for small-scale implementation.

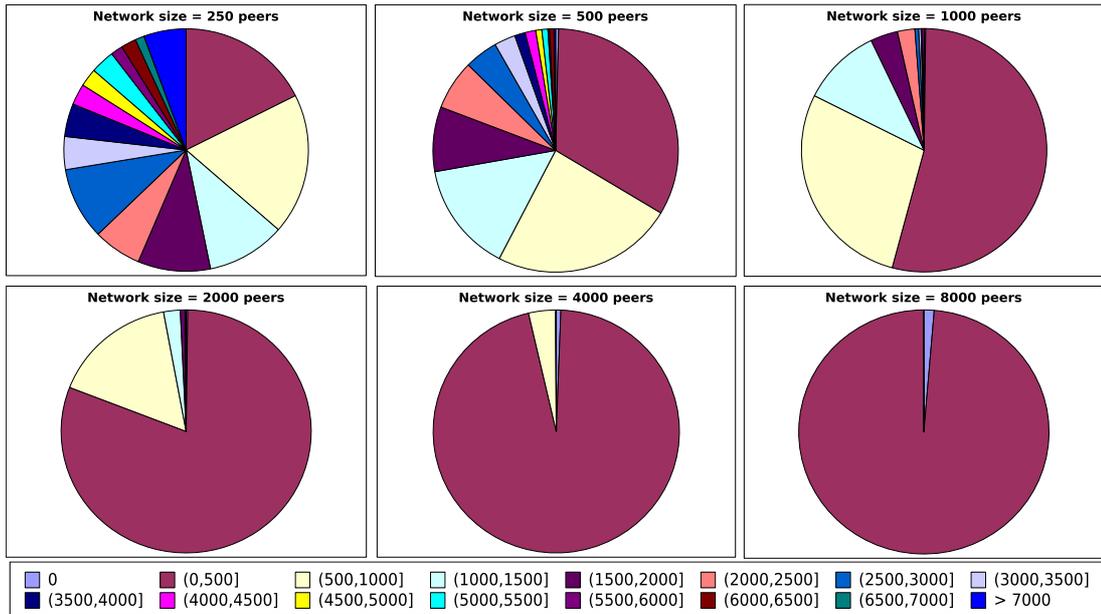


Figure 3.16: Workload (bias elements) distribution throughout the network with an increase in network size in the network-wide P2P-GN implementation.

3.3 Complexity Analysis

In this section, we analyse the message complexity of the P2P-GN’s learning and recall operations, and its recall time complexity with respect to the network growth and compared it with the ensemble k -NN, ensemble ID3, P2P-Cascade SVM, P2P boosting and Ivote-DPV to theoretically prove the scalability of our approach. In our experiments (see Section 4.3), we have evaluated our algorithm within two types of P2P networks: a Chord overlay network and a P2P network with the assumption of optimal connections^{3.14}. We call the former, the P2P-GN with Chord, while, we call the later, the P2P-GN with optimal connections. These two different implementations of the P2P-GN have different communication overhead and processing time since these depend on the network routing performance.

Communication Complexity Here, we present summarised comparison of communication overhead of various distributed P2P classifiers as functions of network size N in Table 3.7. We analysed the communication complexity from two different

^{3.14}Every peer within the network knows the location of each other so that direct connections among them can be established.

views: overall communication complexity at the network level and local communication complexity at the peer level. The overall communication complexity involves the communication overheads in the whole operation (considering the overhead incurred by all participating peers), while, the local communication complexity involves the communication overheads consumed at a single peer. Here, we consider a single peer among the participating peers within the distributed system with the highest communication overheads peer peer for the local communication complexity analysis. During learning, the overall communication overhead of the P2P Cascade

Table 3.7: Summary of the communication complexity as a function of network size (N).

Approach	Overall		Local	
	Learning	Recall	Learning	Recall
Ivote-DPV	*	$\mathcal{O}(N)$	*	$\mathcal{O}(\Omega)$
P2P Cascade RSVM	$\mathcal{O}(N)$	*	$\mathcal{O}(N)$	*
P2P Boosting	$\mathcal{O}(N)$	$\mathcal{O}(N)$	$\mathcal{O}(N)$	$\mathcal{O}(N)$
P2P-GN with Chord	$\mathcal{O}(\log N)$	$\mathcal{O}(\log N)$	$\mathcal{O}(n_H)$	$\mathcal{O}(n_H)$
P2P-GN with optimal connection	$\mathcal{O}(n_H)$	$\mathcal{O}(n_H)$	$\mathcal{O}(n_H)$	$\mathcal{O}(n_H)$
Ensemble k -NN	*	$\mathcal{O}(N)$	*	$\mathcal{O}(N)$
Ensemble ID3	*	$\mathcal{O}(N)$	*	$\mathcal{O}(N)$

n_H refers to the number leaf nodes. Ω refers to the number of immediate neighbours per peer.
 n_H and Ω are independent of N .
 *no communication involved.

RSVM and the P2P Boosting are $\mathcal{O}(N)$ and the local communication overhead is $\mathcal{O}(N)$ since they require models to be exchanged among all sites. However, in the Ivote-DPV and ensemble methods, no communication overhead is consumed during learning since it performs local learning and the aggregation is only performed during recall. This differs from the P2P-Cascade RSVM which performs aggregation during learning phase, but requires no communication during recall phase.

In the ensemble algorithms (ensemble k -NN and ensemble ID3) and the P2P Boosting, every peer sends its local prediction to a requester peer during recall. Hence, for recall, the local and overall communication complexity of these methods are $\mathcal{O}(N)$. In the worst-case, the Ivote-DPV requires $\mathcal{O}(N)$ messages to produce an

exact global classification. Nonetheless, the number of messages per peer is constant $\mathcal{O}(\Omega)$ and independent of the network size. Hence, it is scalable for large networks.

Let n_H be the number of leaf nodes or sub-patterns and n_H is usually smaller than N (n_H is formally given in Equation 3.1 of Section 3.1.2). The P2P-GN in general has the communication complexity per peer that is independent of the network size since every peer communicates with n_H peers. The number of messages per peer in recall process, $M_{recall}(local)$, and the number of messages per peer during learning process, $M_{learn}(local)$ are given in the following Equation 3.6 and Equation 3.5, respectively.

$$M_{learn}(local) = n_H \quad (3.5)$$

$$M_{recall}(local) = 2n_H \quad (3.6)$$

Hence, $M_{learn}(local)$ and $M_{recall}(local)$ are $\mathcal{O}(n_H)$.

The overall number of messages in recall process, $M_{recall}(overall)$, and the overall number of messages in learning process, $M_{learn}(overall)$ are given in the following Equation 3.7 and Equation 3.8, respectively.

$$M_{learn}(overall) = n_H \times M_{LQ} \quad (3.7)$$

where M_{LQ} is the communication overhead (number of hops) for a `LEARN_REQUEST` message to reach a leaf node from a requester.

$$M_{recall}(overall) = n_H \times M_{RQ} + n_H \times M_{RR} \quad (3.8)$$

where M_{RR} is the number of hops for a `RECALL_RESPONSE` message to reach a requester from a leaf node and M_{RQ} is the number of hops for a `RECALL_REQUEST` message to reach a leaf node from a requester.

A sender of a `RECALL_RESPONSE` message knows the requester and it directly sends the response message to the requester. Therefore, M_{RR} is equal to 1 hop. The `LEARN_REQUEST` message and `RECALL_REQUEST` message are lookup messages

which their performance depend on the underlying P2P overlay routing performance. Hence, M_{LQ} and M_{RQ} can be vary; this clearly requires an efficient P2P overlay system.

In the P2P-GN with optimal connections, the lookup overhead is assumed to be 1. Thus, both M_{LQ} and M_{RQ} are also assumed to be 1. Therefore, communication complexity during recall and learning in the P2P-GN with optimal connections is linear with the number of attributes d . The assumption of the P2P-GN with optimal connections is unrealistic in the real world considering the expensive overhead required for a network to satisfy this assumption. However, this measures the best performance of the P2P-GN that can be achieved by improving the routing service. The Chord network uses DHT system. Hence, the communication overhead is affected by the lookup overhead of the DHT-based P2P overlay which is $\mathcal{O}(\log N)$ (particularly the original Chord overlay).

The communication overhead per peer in this approach is independent of the network size. This explains the communication efficiency and its scalability in dealing with a large network. The P2P-GN with Chord requires logarithmic overall communication overhead with the network size and this shows its scalability in dealing with large networks compared to other methods. There are several improvements on the DHT-based system performance that have been made recently, therefore, the performance of our approach, is certainly improving with this development [50, 175, 180].

Recall Time Complexity We focus on the algorithms which perform aggregation during recall (classification) for comparison in this analysis, since the P2P-GN belongs to same category. The summary of overall recall time complexity is give in Table 3.8. The P2P Boosting and ensemble methods (ensemble k -NN and ensemble ID3) perform sequential reduction of local predictions from N peers, while, the Ivote-DPV performs iterative combining process which requires N iterations at the worst-case. Since the P2P-GN uses parallel processes (lookups and reduction), its recall time is better than the Ivote-DPV, P2P Boosting and ensemble methods. Although the lookups are executed in parallel, a single lookup itself requires a

Table 3.8: Summary of the recall time complexity as a function of network size (N)

Approach	Recall Time Complexity
Ivote-DPV	$\mathcal{O}(N)$
P2P Boosting	$\mathcal{O}(N)$
P2P-GN with Chord	$\mathcal{O}(\log N)$
P2P-GN with optimal connections	$\mathcal{O}(1)$
Ensemble k -NN	$\mathcal{O}(N)$
Ensemble ID3	$\mathcal{O}(N)$

number of hops before the lookup message reaches the target peer. Therefore, for the P2P-GN with Chord implementation, the recall processing time is dominated by the lookup time which is logarithmic with the network size. The P2P-GN with optimal connections, however, is not affected by the lookup time. Hence, the recall processing time grows constant with the network size.

Limitation when $d \geq N$ — We observe that the communication overhead of the P2P-GN with Chord can be higher than the overhead incurred by the other methods when the number of attributes d is greater than the network size N (note that n_H is at most b). However, this rarely occurs since the size of the P2P networks are usually large.

3.4 Summary

The growth in network size brings the following issue to the P2P-based distributed pattern recognition computing: the amount of communications and time to produce an exact global classification usually grows linearly with an increase in network size. The gossip-based or local algorithms that operating within a large network, usually converge to the exact global classification in a high number of iterations and this incurs a high magnitude of communications. These iterations and communications are required for peers to share and combine their local models and local results

among themselves. The proposed algorithm brings an exact, single-cycle and fully-distributed solution for pattern recognition within large P2P networks. The key findings in this chapter are listed as follows:

- The P2P-GN algorithm provides an exact global classification algorithm within large networks with comparable accuracy to the centralised, state-of-the-art algorithms (ID3, k -NN, naive Bayes, 1-NN, RBFN and BPNN). This has been validated in the experimental results on four datasets with a large number of attributes (i.e., Arcene, ISOLET, MNIST and MFeat).
- We have demonstrated that the communications per peer in the P2P-GN is independent of the network size in a series of experiments with varying network sizes. Hence, it is scalable for large-scale distributed networks as defined in Definition 2.2^{3.15} of Section 2.
- The communication overhead of the P2P-GN implementation on a network with optimal connections grows in a constant rate, while, the P2P-GN implementation within a Chord overlay network grows logarithmically with an increase in network size. This shows that the communication overhead of the P2P-GN can be reduced significantly with the improvement in the P2P lookup performance.
- We have experimentally demonstrated that the communication overhead and recall time of the P2P-GN which is implemented within a Chord overlay network are lower than the Ivote-DPV and ensemble ID3 since it uses a single-cycle operation. In every learning or recall process in the P2P-GN, a requester sends out n_H messages and receives n_H responses where n_H is independent of the network size^{3.16}. Thus, the communication overhead remains low despite the increase in network size. The recall time is also low because it uses a highly parallel processes, in addition its single-cycle operation.

^{3.15}A distributed algorithm is network-scalable if the communication per peer is independent to the network size.

^{3.16} n_H refers to the number of leaf nodes or the number of sub-patterns.

- Compared to other distributed algorithms (i.e., Ivote-DPV, ensemble k -NN, ensemble ID3 and ensemble naive Bayes), the P2P-GN remains accurate despite operating within imbalanced class distribution in the following scenarios: imbalance class distribution and small local dataset size.
- In our experiments, we have shown that the use of DHT balances the workload among peers within the network, particularly when the network is large. This demonstrates the scalability of our approach for large networks.

The lack of synchronisation requirement in our scheme allows it to work correctly within large asynchronous environment such as P2P networks. Moreover, it is scalable for large networks—communications per peer in the P2P-GN is independent of the network size. It does not require a highly iterative process, it is highly parallel and has a low aggregation communication overhead. The main advantage of this approach is it requires only one iteration (single-cycle operation); the number of iterations reflects the communication delay for the overall computation where a higher number of iterations involves a greater communication delay—effectively magnifying the response time. This allows an exact global classification able to be provided with a fast response time for ad hoc query.

The simulation results also show that our method is more communication-efficient than the Ivote-DPV and other semi-distributed ensemble classifiers. The continuous improvements of the routing performance in the DHT-based overlay network in the recent studies will improve the communication efficiency and time efficiency of our method since our algorithm relies on the DHT-based overlay networks to link the bias elements [50, 175, 180]. The use of DHT technology also enables our algorithm to be adaptive to the different network scale and this permits the use of this algorithm for grid P2P or large-scale P2P environments.

Since we build a single global classifier which represents all datasets from the whole network, our algorithm is invariant to the imbalanced data distribution whereas

the Ivote-DPV, ensemble k -NN, ensemble naive Bayes and ensemble ID3 fail to preserve their accuracy when the local dataset is small or have imbalanced class distribution. Additionally, the use of two stage prediction in the P2P-GN also contributes to avoid the problem with imbalanced class distribution which is a major issue in the P2P data mining. Ang et al. [9] may resolve this issue but at a cost of expensive communication overhead which only suit a small-scale P2P network.

A drawback of this approach is that a large number of attributes may cause a high computational overhead at a requester peer when dealing with a high-dimensional data to aggregate these local predictions. Since the complexity of the P2P-GN is linear with the number of attributes, it is expected that the computational overhead is $\mathcal{O}(N)$ when the network size (N) is smaller than the number of attributes (d). However, in practice, N is always much greater than d since a P2P network may involve hundred thousands to millions of peers. Next, we will show how we encounter this problem in Chapter 4.

Chapter 4

A P2P Classification Algorithm for Large Datasets

A growing number of studies with distributed classification suggest the efficiency and potential of distributed methods in dealing with scalability problem [183, 41, 140]. However, most of the distributed classification algorithms which are built for this purpose are cluster-based methods (e.g., ensemble classifiers and artificial neural networks (NN)) that have a coordinator to vertically^{4.1} or horizontally^{4.2} partition the large datasets into several small subsets; these small subsets are then assigned to the distributed components for processing within the cluster. In contrast, peers within P2P networks collect their own local data; perform local classification using the data; and an aggregation is used to obtain a global classifier or a global prediction in collaboration with other peers. Such distributed classification techniques are effectively similar to the horizontal partitioning, however, they have no centralised entity to control the partitioning process—unlike the cluster-based methods where there is a coordinator to control the amount of data per partition or to add more resources to the system.

With the rapidly growing amount of data, the size of local datasets is also increasing. Given that most of the available pattern classification algorithms are

^{4.1}Splitting instances from a whole dataset; every instance has all features in the feature space.

^{4.2}Splitting feature space into multiple sets; every site has equal number of instances but with different feature set.

usually computational-demanding, this burdens the peers and may discourage their participation. P2P networks are usually formed by the availability of participants willing to provide resources (CPU cycle, storage) and given the fact that the use of mobile devices with limited resources is prevalent within P2P networks today, it is crucial to ensure the computational overheads at these peers remain low, despite the increasing dataset size.

Despite this, most of the available P2P classification algorithms focus on improving the scalability for large networks while neglecting the scalability issues related to the dataset factor [54, 125, 17, 66]. The high computational requirement of the available algorithms restricts the ability of the system to scale with high-dimensional and large datasets. For instance, every local SVM classifier in the P2P Cascade SVM requires an expensive optimisation process and every local Ivote classifier in the Ivote-DPV converges to an accurate model in several iterations. The P2P-GN, however, has taken a different way, it builds a single classifier in collaboration of all peers within the network instead of having every peer builds its own classifier. This is achieved by distributing the light-weight and fine-grained components across the network. Therefore, the computation task is effectively distributed to all peers. As the memories in a P2P environment are frequently updated, the online learning and the single-cycle learning characteristics of this algorithm provide a fast learning process for large datasets and ensure its computational complexity remains low.

During learning, the high data dimension has an effect on the computation overheads at a requester^{4.3}, while, it has not effect on the computation overheads at a leaf node. The communication overheads at a requester during learning may be large for a high-dimensional problem. However, these overheads can be easily capped by distributing the querying segments in a spanning tree so that a peer only communicates with at most m other peers. Therefore, we shall not discuss this in this chapter. Instead, the similar learning procedure as described in Chapter 3 is used here.

^{4.3}A requester breaks a feature vector into multiple segments and perform identifier generation.

Unlike the P2P-GN learning phase which involves no aggregation, the recall (classification) phase incurs expensive aggregation overheads to combine local predictions from a large number of peers for a high-dimensional problem. Therefore, we propose an approach to deal with high-dimensional problems in order to maintain a small computational overhead per peer. In the proposed convergecast recall, the large computational burden is divided among several peers which form a tree structure. These peers work in parallel where each peer only incurs a small computation overhead. Figure 4.1 summarises the procedures in the P2P-GN which involve a learning procedure and two types of recall methods. The flat recall refers specifically to the recall method in the P2P-GN which has been presented in Chapter 3, while, the convergecast recall will be introduced next in this chapter.

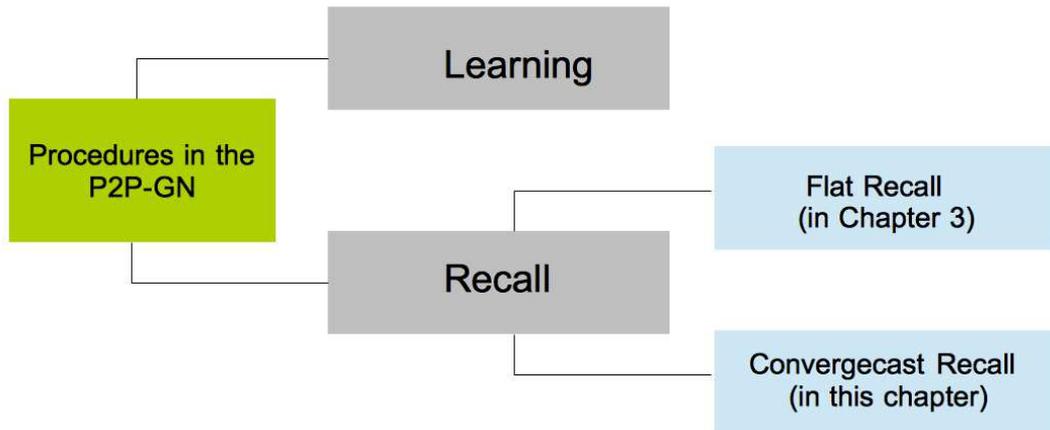


Figure 4.1: The P2P-GN procedures involves a learning and two types of recall methods: the flat recall and the convergecast recall. The flat recall refers to the recall method that has been described in Chapter 3, while, the convergecast recall refers to the method which will be introduced later in this chapter.

In addition to the large datasets issue, we also consider the reliability aspect of the P2P-GN in dealing with network churn. Churn is defined as random disruptions over the network which are caused by the events of peers joining and departure; it is a major issue in P2P networks. Basically, the joining peers give a small impact on P2P systems as the data are quickly recovered as soon as a peer joins a network. However, peers may leave the network gracefully or unexpectedly. Graceful peers

departure give less impact on the system compared to unexpected failure since during the graceful departure, these peers will notify their neighbours before leaving the network. Thus, these neighbours are well-prepared and able to quickly recover the data. Unexpected peers departures impact the system the most as the failure can only be detected at the next stabilisation (routing table updates) and we may lose the important data if there is no replica of the data anywhere else in the network. This may also result in degrading the accuracy of the P2P-GN.

The main aim in this chapter is to empirically evaluate the efficiency of the P2P-GN algorithm that we have proposed in the Chapter 3 for a scalable distributed pattern classification algorithm for large datasets. We will propose the convergecast recall approach to reduce the aggregation overhead of the P2P-GN at the requester peer ^{4.4} in dealing with high-dimensional datasets. Additionally, we will also evaluate the reliability of the P2P-GN and introduce a fault-tolerant management to support the correctness of the algorithm within dynamic networks. Then, this chapter will examine the effectiveness of proposed convergecast recall approach and the fault-tolerant management.

In Section 4.1, we present the convergecast recall of the P2P-GN for high-dimensional problems. Here, we also show how we reduce the aggregation overhead at a requester peer and explain the in-network convergecast recall method. Then, in Section 4.2, we perform a comparative analysis of the communication overheads during learning and recall in the P2P-GN, HGN, DHGN and BPNN. This involves the comparison of the two recall approaches (flat recall and convergecast recall) in the P2P-GN. We further demonstrate the performance of the P2P-GN during learning and recall in dealing with large datasets through experiments in Section 4.3. Then, we discuss the complexity analysis of these recall approaches and the learning complexity of the P2P-GN in Section 4.4. In Section 4.5, we describe the dynamic network problem in the P2P-GN and this is followed by the description of our proposed fault-tolerance approach to deal with this problem in Section 4.6. The

^{4.4}a requester peer is a peer that submits a query and aggregates results.

proposed fault-tolerance approach is then evaluated with regards to its effectiveness and efficiency in Section 4.7. Finally, the conclusion in Section 4.8 summarises the key ideas and achievements in this chapter. Note that the notations that are used in this chapter can be referred in Tables A.2 of Appendix A.

4.1 The Convergecast Recall for High-Dimensional Problems

This section explains our solution to maintain the scalability for high-dimensional problems. We discuss on how a tree structure can be constructed within the network to leverage the workload among several peers. Here, a number of peers work together during recall to reduce the aggregation overhead at a single peer. Then, we describe the P2P-GN recall process by using a convergecast aggregation approach. Note that the convergecast approach only involves the recall phase since the learning phase in the P2P-GN does not require any aggregation. Therefore, the storage structure of the convergecast approach equals that of the flat approach; this is because the internal nodes do not store any memory, they only forward the recall requests and perform an intermediate aggregation. There are two main steps in this approach: first is the spanning tree propagation and second is the convergecast aggregation.

(i) Spanning tree propagation. In the first step, a spanning tree is build to propagate the querying bias elements. The process is started when a requester node needs to send a large number of queries for the bias elements (the number of queries is larger than m) where the maximum number of connections is m . It divides the querying bias elements into m subsets and send these subsets to a number of peers. The receiving peer then decides whether the number of received queries is still larger than m ; if so, it further divides the queries into m subsets and assigns these subsets to other peers. This forms a tree structure within the network with the requester peer as a root node and other participating peers (except leaf nodes) are internal

nodes. Let the root node be at level 0, the leaf node be at level \bar{h} where \bar{h} is the height of the spanning tree.

(ii) Convergecast aggregation. In the second step, the local results from the leaf nodes are sent to the internal peers that forward the queries. The peers aggregate the results and send the aggregated results to their parent nodes. These processes are done iteratively until the aggregated results reach the root node.

These two steps in the convergecast approach are explained further in the following.

4.1.1 Tree Construction

The tree construction starts from a requester as a root. The computational load per peer is maintained by growing the tree. This increases the scalability and efficiency of the algorithm in handling high-dimensional datasets. This approach also improves the computation time as the processes are performed in parallel at each level. Here, we assume that every peer has an equal computational capability and we define the limit m on the maximum number of children that a peer can handle. m may also refer to the maximum number of simultaneous connections a peer can have (see Table C.1 of Appendix C). We also provide an additional information on the tree construction, including the pseudo code to build this tree in Algorithm C.1 of Appendix C.

Lets $\widehat{X} = \{\hat{x}\}_{i=1}^{n_H}$ be an array of sub-patterns where n_H is the total number of sub-patterns/leaf nodes and each sub-pattern has d_s number of attributes. Every sub-pattern \hat{x}_i is processed by a leaf node $h(i)$. We have previously described the process to create these sub-patterns/leaf nodes from a feature vector \mathbf{x} with data dimension d in Section 3.1.2 of Chapter 3. The parameter OV equals the number of overlapping positions of two sub-patterns and $OV = 0$ suggests that these sub-patterns are disjoint. The initial tree structure is a 1-height tree consisting of n_H leaf nodes and a root node. In reducing the workload (the leaf nodes) at a single

node, vertices are added with their edges connecting to the parent node and the workload is distributed among these vertices. Starting from a root, the processes of adding vertices and dividing the workload are executed successively until reach the limit on the amount of workload to be handled by a single peer—given that each node has a constraint on m which is the maximum number of children of each parent can have.

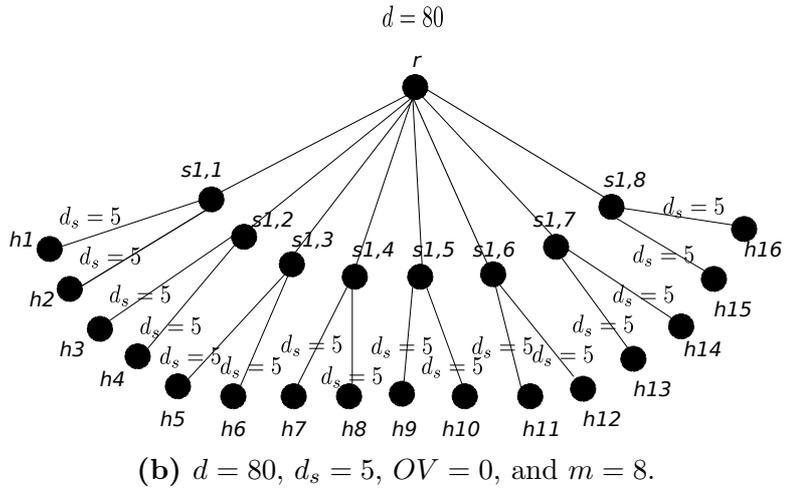
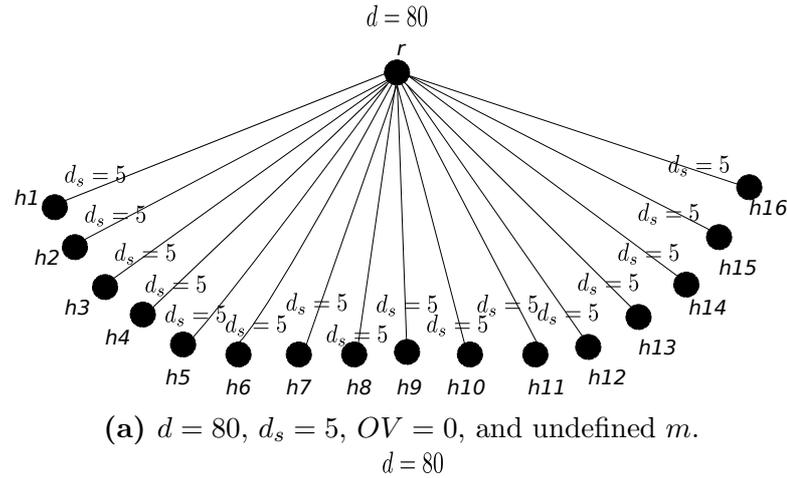


Figure 4.2: Two examples of the convergecast recall tree structure for $d = 80, d_s = 5$, and $OV = 0$ (a) with undefined m , and (b) with $m = 8$. The label r refers to a root node, $h(i)$ refers to the i th leaf node and $s(j, i)$ refers to an i th internal nodes at level j .

Figure 4.2a shows an example tree structure for a classification problem with $d = 80, d_s = 5, OV = 0$, and undefined m . The height of the resulting tree is denoted by \bar{h} and the total nodes is denoted by n_R . The resulting structure has \bar{h} equals 1, n_R equals 17, and the root node has $n_H = 16$ children. Figure 4.2b shows an example structure when $d = 80, d_s = 5, OV = 0$, and number of children $m = 8$.

As the initial number of leaf nodes is 16 which is larger than $m = 8$, the tree can then be expanded. Hence, $m = 8$ children are created at the root node. These children become internal nodes where leaf nodes are assigned using greedy approach to them. Thus, at level 1, we have eight internal nodes and each of these internal nodes are responsible for two leaf nodes. The resulting structure has \bar{h} equals 2 and n_R equals 25. The reducing task for a distributed structure in Figure 4.2a requires 16 sequential steps to combine all 16 local results at the root node, while, the structure in Figure 4.2b just requires nine sequential steps in total. This includes one step at nodes in level 1 and eight steps at the root node (level 0).

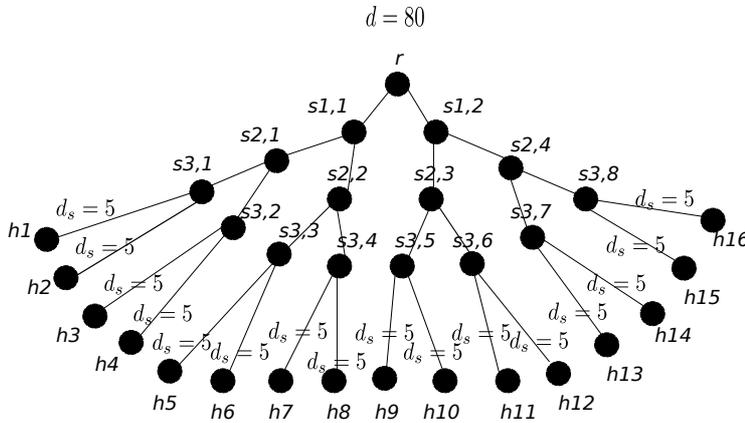


Figure 4.3: An example of a convergecast recall tree structure for $d = 80$, $d_s = 5$, $OV = 0$, and $m = 2$. The label r represents a root node, $h(i)$ refers to the i th leaf node, and $s(j, i)$ refers to an i th internal nodes at level j . The resulting tree \bar{h} is 4 and $n_R = 31$.

Figure 4.3 shows the tree structure when m value is two. The workload is divided successively until the number of leaf nodes that is connected per node is equal or less than $m = 2$ as we can see at the lowest level. The number of leaf nodes is divided with $m = 2$ and the division product is 16 which is greater than $m = 2$. Therefore, $m = 2$ children is created with eight leaf nodes are assigned to each child respectively. As the number of leaf nodes being assigned to these children are still greater than m , these children then become internal nodes. Then, they are ready to create their child nodes and distribute their workload to these nodes. The same process is successively executed until the magnitude of leaf nodes per internal node

is not greater than $m = 2$. The complete binary tree structure in Figure 4.3 requires four sequential steps to combine all 16 local results.

The number of nodes in a convergecast tree is given in Equation 4.1.

$$\begin{aligned}
 n_R &\leq \text{the number of internal nodes plus the root node} + & (4.1) \\
 &\quad \text{the number of leaf nodes} \\
 &\leq \sum_{l=0}^{\bar{h}-1} m^l + n_H \\
 &\leq \frac{m^{\bar{h}} - 1}{m - 1} + n_H
 \end{aligned}$$

where \bar{h} is the height of the tree, $\bar{h} \leq \log_m n_H$ and n_H is the number of leaf nodes (refer to Equation 3.1 of Chapter 3). The bound of n_R is given in Property 4.1.

Property 4.1. *Given n_H , the lower bound of n_R is equal to n_H , while, the upper bound of n_R is $(2n_H - 1)$. The value of n_R is bounded by the number of attributes (data dimension), d .*

Proof. The largest n_R is found when m is the smallest, while, the smallest n_R is found when m is the largest. The smallest value of m is 2. Hence, if $m = 2$, then n_R is as follows.

$$\begin{aligned}
 n_R(m = 2) &= \frac{2^{\log_2 n_H} - 1}{2 - 1} + n_H \\
 &= 2n_H - 1
 \end{aligned}$$

The largest value of m is n_H . If $m = n_H$, then n_R is as follows.

$$\begin{aligned}
 n_R(m = n_H) &= \frac{n_H^{\log_{n_H} n_H} - 1}{n_H - 1} + n_H \\
 &= \frac{n_H - 1}{n_H - 1} + n_H \\
 &= n_H
 \end{aligned}$$

Hence $n_H \leq n_R \leq 2n_H - 1$. Since n_H is $\mathcal{O}(d)$, then n_R is also $\mathcal{O}(d)$. \square

4.1.2 Convergecast Recall

The convergecast recall requires the selection of peers as internal nodes. The peers can be elected according to various factors such as their uptime duration, trust level,

latency and the hardware performance. The studies regarding peer selection—to choose one or more peers which are able to hold a more important role than other peers—have been widely conducted and various methods have been proposed from the literature [149, 78, 146]. Therefore, we omit this subject in this thesis and continue with an assumption of an election function $\text{elect}(m)$ which exists for every peer to elect m peers to be its children.

The convergecast recall procedures at a requester, an internal node and a leaf node are described separately in Algorithms 4.1, 4.2 and 4.3, respectively. There are three main functions in these procedures: (i) $\text{localRecall}(\cdot)$, (ii) $\text{agg}(\cdot)$ and (iii) $\text{finalPrediction}(\cdot)$. These functions have been described previously in Section 3.1.8 of Chapter 3. Two types of messages: `FORWARD_REQUEST` and `FORWARD_RESPONSE`, are introduced in the convergecast recall; these are defined as follows.

Definition 4.2 (`FORWARD_REQUEST`($\{\hat{S}\}$)). *A `FORWARD_REQUEST`($\{\hat{S}\}$) message is sent from a node i to a node j to request the node j to send `RECALL_REQUEST` messages for a set of bias identifiers $\{\hat{S}\}$ on its behalf.*

Definition 4.3 (`FORWARD_RESPONSE`). *A `FORWARD_RESPONSE` message is sent by internal node i to the parent node j ; it contains the intermediate aggregation result.*

The process is started at a requester which generates n_H bias identifiers from the n_H input sub-patterns (see Algorithm 4.1). If n_H is less than or is equal to m , it sends `RECALL_REQUEST` messages with the querying bias identifiers, directly. Otherwise, the requester peer divides the bias identifiers into m sets, then it sends a `FORWARD_REQUEST` message to a selected peer $p \in \mathfrak{P}$ for each of these sets. \mathfrak{P} consists of a set of peers that are selected using $\text{elect}(\cdot)$ function, a `FORWARD_REQUEST` message is sent for each peer $p \in \mathfrak{P}$.

After receiving a `FORWARD_REQUEST` message which contains \hat{S} , a peer (an internal node) then decides whether to stop the division process (if the number of bias identifiers in \hat{S} is smaller or equal to m) or to proceed with the division process (if the number of bias identifiers in \hat{S} is greater than m). This is summarised in Line

Algorithm 4.1. Convergecast P2P-GN Recall at a requester peer p

```

1: Input:  $\{\hat{x}_i\}_{i=1}^{n_H}$  (input sub-patterns) and  $m$ .
2:
3: for all  $\{\hat{x}_i\}_{i=1}^{n_H}$  do
4:   Generate a bias identifier —  $\psi(i) \leftarrow \mathbf{q}(\hat{x}, i)$ .
5: end for
6: if  $n_H > m$  then
7:   Divide the bias identifiers  $\{\psi(i)\}_{i=1}^{n_H}$  into  $m$  sets,  $\{\hat{\mathbf{S}}_i\}_{i=1}^m$ 
8:    $\mathfrak{P} \leftarrow \text{elect}(m)$ .
9:   for all  $\{\hat{\mathbf{S}}_i\}_{i=1}^m$  do
10:    Send FORWARD_REQUEST for  $\hat{\mathbf{S}}_i$  to  $p_i \in \mathfrak{P}$ .
11:   end for
12: else
13:   Send RECALL_REQUEST for  $\{\psi(i)\}_{i=1}^{n_H}$ .
14: end if
15:
16: upon receiving RECALL_RESPONSE or FORWARD_RESPONSE messages
17: Receive  $\{\hat{r}(p_i)\}_{p_i \in \mathfrak{P}}$  predictions from all peers in  $\mathfrak{P}$ .
18: Aggregate prediction  $\hat{R}(G) = \text{agg}(\{\hat{r}(p_i)\}_{p_i \in \mathfrak{P}})$ .
19:  $\mathcal{L} = \text{finalPrediction}(\hat{R}(G))$ .

```

3 of Algorithm 4.2. In the former case, the peer sends a RECALL_REQUEST messages for lookup. While in the latter, it splits $\hat{\mathbf{S}}$ into another m subsets and then sends these subsets to another m peers through a FORWARD_REQUEST messages.

Upon receiving a RECALL_REQUEST query for ψ , a peer b (a leaf node) verifies that ψ is within its subspace, i.e., $b.ID \leq \psi < b_{suc[0]}.ID$ (see Line 2 of Algorithm 4.3). Then, it searches for ψ from its local storage. If found, then b performs the `localRecall(ψ)` function and forwards the prediction output to the parent through a RECALL_RESPONSE message. Subsequently, at a parent node (who receives the RECALL_RESPONSE message), the predictions $\{\hat{r}(j)\}_{j=i}^m$ from its m children (\mathfrak{P} who receive its RECALL_REQUEST messages) are aggregated using `agg(.)` (see Line 13 of Algorithm 4.2). The aggregated prediction, \hat{r} is forwarded to the next level using a FORWARD_RESPONSE message.

Upon receiving the FORWARD_RESPONSE messages, an internal node performs an intermediate aggregation and sends the aggregated prediction to its parent node. The intermediate aggregations are performed in parallel for all nodes within the same level. The rationale of this action is to reduce the computational cost at the

Algorithm 4.2. Convergecast P2P-GN Recall at internal node s

```

1: upon receiving  $\hat{S}$  in FORWARD_REQUEST message.
2: if  $|\hat{S}| > m$  then
3:   Partition the set  $\hat{S}$  into  $m$  subsets,  $\{\hat{S}'_i\}_{i=1}^m$ 
4:    $\mathfrak{P} \leftarrow \text{elect}(m)$ .
5:   for all  $\{\hat{S}'_i\}_{i=1}^m$  do
6:     Send FORWARD_REQUEST for  $\hat{S}'_i$  to  $p_i \in \mathfrak{P}$ .
7:   end for
8: else
9:   Send RECALL_REQUEST for  $\{\psi(i)\}_{i=1}^{|\hat{S}|}$ .
10: end if
11:
12: upon receiving RECALL_RESPONSE or FORWARD_RESPONSE messages
13: Receive  $\{\hat{r}(p_i)\}_{p_i \in \mathfrak{P}}$  predictions from all peers in  $\mathfrak{P}$ .
14: Aggregate prediction  $\hat{r}' = \text{agg}(\{\hat{r}(p_i)\}_{p_i \in \mathfrak{P}})$ .
15: Send  $\hat{r}'$  to the parent through FORWARD_RESPONSE message.

```

Algorithm 4.3. Convergecast P2P-GN Recall at a leaf node b

```

1: upon receiving a query for  $\psi(i)$  from a peer  $s$ 
2: if  $b.ID \leq \psi < b_{suc[0]}.ID$  then
3:   if bias element ( $\psi$ ) is found then
4:     Calculate prediction  $\hat{r} = \text{localRecall}(\psi)$ .
5:   else
6:      $\hat{r} = \{\text{"unknown"}, 0, 0\}$ .
7:   end if
8:   Send the RECALL_RESPONSE( $\hat{r}$ ) to the peer  $s$ .
9: else
10:  Forward to another peer by using routing service.
11: end if
12:

```

root node (the requester) by sequentially aggregating the local results from leaf nodes as practised in the flat recall. Finally, the aggregation at the root node produces a global prediction $\hat{R}(G)$ and the final prediction is then determined in function `finalPrediction(.)` execution (refer to Section 3.1.8 of Chapter 3) as described in Line 19 of Algorithm 4.1.

The aggregated vote, $agg\text{-}v(c)$ for option c represents the number of leaf nodes that predict the queried pattern as an option c . All nodes basically knows the number of leaf nodes in their sub-tree. Therefore, for instance, when an identifier is found at a node b with a score table which has class c , then the maximum value for

$agg_v(c)$ equals $n_H(b)$ where $n_H(b)$ is the number of leaf nodes within the sub-tree of the node b . An example of the convergecast aggregation steps is as follows:

Example

An example of the convergecast aggregation step for the convergecast recall with four leaf nodes and $m = 2$ is given in Figures 4.4, 4.5 and 4.6.

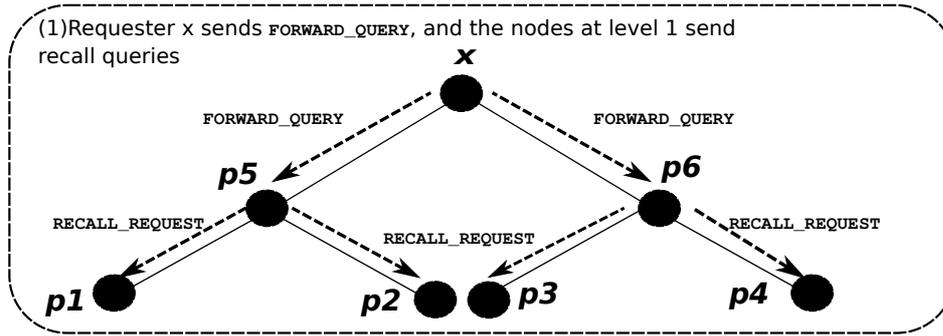


Figure 4.4: Step 1: A two level spanning tree is created during convergecast recall process.

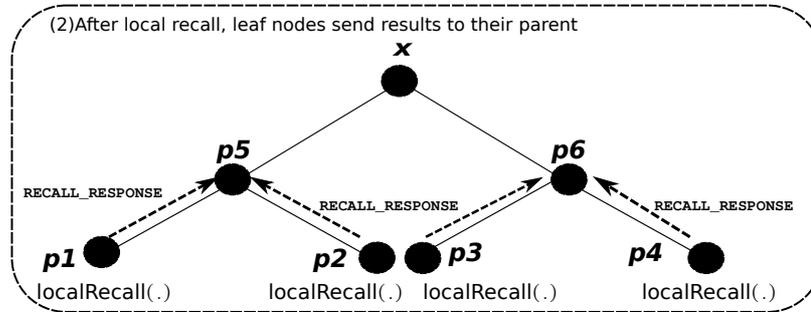


Figure 4.5: Step 2: The local predictions from leaf nodes p_1 , p_2 , p_3 and p_4 are sent to the internal nodes p_5 and p_6 .

A spanning tree is created with two nodes at the first level and four nodes at the second level. The `FORWARD_REQUEST` messages are sent by the requester to two selected peers (internal nodes): p_5 and p_6 ; these two peers then send the `RECALL_REQUEST` messages to four peers (leaf nodes) $\{p_1, p_2, p_3, p_4\}$. After executing `localRecall`, the leaf nodes response to the requests with `RECALL_RESPONSE` and the internal nodes then aggregate then outcome and forward the response back to

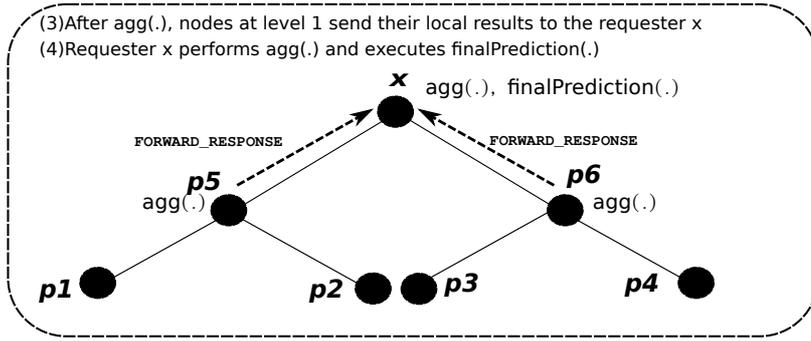


Figure 4.6: Step 3 and 4: The intermediate aggregation results from internal nodes $p5$ and $p6$ are forwarded to the requester x .

the requester using `FORWARD_RESPONSE`. The outputs from the $p5$ and $p6$ are then aggregated at the root node peer x (the requester) which produces a final prediction \mathcal{L} using the function $\text{finalPrediction}(\cdot)$

4.2 Evaluating Communication Efficiency: Comparative Analysis

We performed comparative analysis of the P2P-GN with the Backpropagation Neural Network (BPNN), Hierarchical Graph Neuron (HGN) [140] and Distributed HGN (DHGN) [138] on the communication efficiency. As in the P2P-GN, the architecture of the BPNN, HGN and DHGN depend on the data dimension (number of attributes). Hence, we show the communication efficiency of our algorithm compared to these algorithms by analysing the local communication overheads at the peer level and overall communication overheads at the network level with respect to the data dimension. The number of messages for every learning procedure and recall procedure is used as a metric to evaluate our algorithm in this study.

Analysis Set-up

We simulated the HGN, BPNN, P2P-GN and DHGN for the following classification problem. Let u be the size of domain (e.g., 2 for binary data $\{0, 1\}$), \mathcal{C} be the set

of outputs (classes) and d be the data dimension. The classification problem has the following characteristics: the input domain is $\{a, b, c\}$ ($u = 3$), the outputs are $\mathcal{C} = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$ and we varied d within a range of 100 to 700.

The HGN has a fixed architecture (depends on the data dimension), while, the architectures of BPNN, P2P-GN and DHGN can be tuned based on the implementation [140]. Therefore, unlike BPNN, P2P-GN and DHGN, the HGN does not require any parameter setting. The parameter setting for these algorithms are obtained by manual tuning and they are given as follows. We set the sub-pattern size, d_s in the DHGN and P2P-GN to $\lceil \log_2 d \rceil$. By doing so, we can ensure that d_s grows sufficiently small (relative to d). For the P2P-GN, we set $OV = \lceil \frac{d_s}{2} \rceil$ and $m = 10$. The value $m = 10$ can be considered low in a real P2P system (see Table C.1 of Appendix C on the best selection of m based on a real world P2P application.). Note that a smaller m requires a higher number of messages in the convergecast recall (see Property 4.1). In this analysis, we used three-layer BPNN where the hidden layer consists of $\frac{|\mathcal{C}|+d}{2}$ nodes. Further discussion on the communication overheads and the resulting architectures of the BPNN, HGN, DHGN and P2P-GN (including the tree structure for convergecast recall) in this analysis are provided in Appendix D.

Evaluating Local Communication Overheads At The Peer Level

For each algorithm, we selected a peer which incurs the highest number messages during learning and recall to observe the peer level communication overheads. The observed peers are as follows.

- BPNN : a hidden neuron.
- DHGN : the *S&I* node (base station) .
- HGN: the *S&I* node (base station).
- P2P-GN: a requester node.

The number of messages per instance at the observed peers are shown in Table 4.1. The local communication overhead in the HGN is the highest (quadratic as a function of d), while, the overhead in the P2P-GN is the lowest (linear as a function of d). This is because in the HGN, $\mathcal{O}(d^2)$ active nodes communicate with their immediate neighbours and $S\&I$ in each learning and recall session.

Table 4.1: The maximum number of messages per peer during learning and recall with varying d values.

d	Learning				Recall				
	BPNN	DHGN	HGN	P2P-GN	BPNN	DHGN	HGN	P2P-GN	
								Flat	Convergecast
100	1,200	315	10,000	32	110	315	10,000	64	20
200	2,200	625	40,000	49	210	625	40,000	98	20
300	3,200	934	90,000	74	310	934	90,000	148	20
400	4,200	1,245	160,000	99	410	1,245	160,000	198	20
500	5,200	1,556	250,000	124	510	1,556	250,000	248	20
600	6,200	1,860	360,000	119	610	1,860	360,000	238	20
700	7,200	2,170	490,000	139	710	2,170	490,000	278	20

However, in the DHGN, the active nodes in the subnets do not send their indices to the $S\&I$, instead, only active nodes at the top layer of all subnets send their indices to the $S\&I$. This results in less communication overhead; it grows linearly as a function of d . The communication overheads of the HGN and DHGN (in learning and recall operations) are equal since they perform similar processes.

The number of messages at a requester during recall in the flat recall of the P2P-GN are double the number of messages during learning since in recall, the recall requests are sent to leaf nodes and these leaf nodes response with their local predictions to the requester. In the convergecast recall, a constant local communication overhead is incurred by a requester peer. This is because the number of messages per peer is bounded by m parameter that specifies the number of maximum child nodes per parent node (in a convergecast recall, a requester peer is also a root node of the convergecast tree). The number of messages at the requester in learning and in flat recall are smaller when the data dimension is 600 than when the data dimension is 500 since n_H is 124 when the data dimension is 500 and 119 when the data dimension is 600. This is because n_H is equal to $\left\lceil \frac{d-d_s}{d_s-OV} + 1 \right\rceil$; and

we set $d_s = \lceil \log_2 500 \rceil = 9$ and $OV = \lceil \frac{9}{2} \rceil = 5$ when the data dimension is 500; and $d_s = \lceil \log_2 600 \rceil = 10$ and $OV = \lceil \frac{10}{2} \rceil = 5$ when the data dimension is 600. .

In BPNN, every hidden neuron receives d inputs from the input layers and it then sends $|\mathcal{C}|$ outputs from hidden layer to the output layer. During backpropagation (in learning), $|\mathcal{C}|$ output neurons propagate their error values to every hidden neuron (at the immediate layer to the output layer). Hence, the communication overhead in the BPNN is higher during learning compared to during recall because of the propagation phase and the iterative learning.

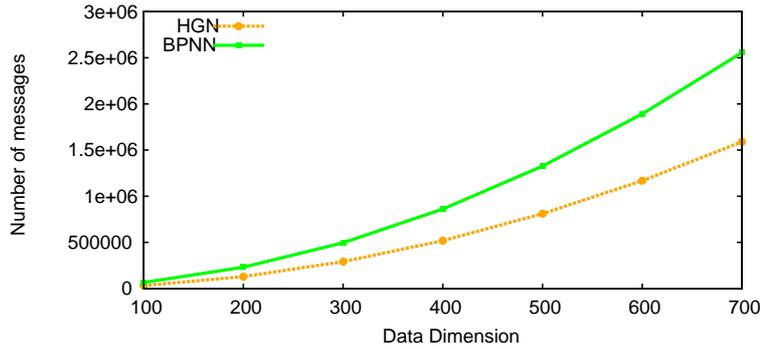
This analysis demonstrates the communication efficiency of the P2P-GN compared to the BPNN, HGN and DHGN. It also demonstrates that the convergecast recall is highly scalable for high-dimensional datasets where it requires constant number of messages per peer.

Evaluating Overall Communication Overheads At The Network Level

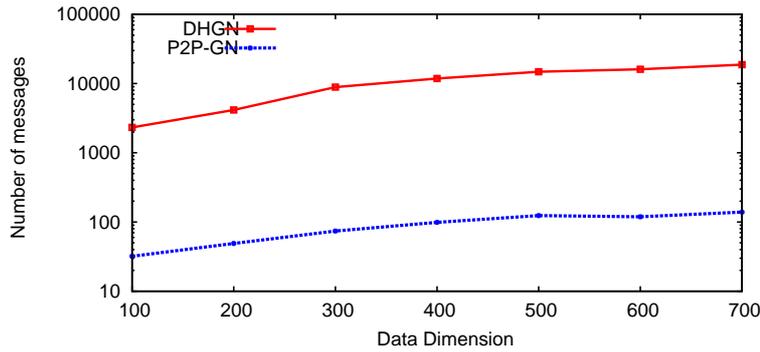
In the simulations, we computed the number of messages which occurred during learning and recall. For each of the algorithm, we evaluated the communication overheads with an increase in data dimension by calculating the number of messages during learning of a training instance ($q = 1$) and during recall of a test instance.

The number of messages in the HGN and BPNN are significantly higher than the P2P-GN and DHGN. Therefore, to show the growth of communication overhead in these algorithms with an increase in data dimension, we plot the results in two different graphs: Graph 1 presents the number of messages in the HGN and BPNN (see Figures 4.7a and 4.8a) and Graph 2 presents the number of messages in the DHGN and P2P-GN (see Figures 4.7b and 4.8b).

The message complexity during learning in the P2P-GN and DHGN are linear as a function of data dimension, while, the message complexity in the HGN and BPNN are quadratic. The number of messages in the HGN is significantly large compared to the P2P-GN and DHGN, but, it is smaller than the BPNN.



(a) Graph 1: BPNN and HGN

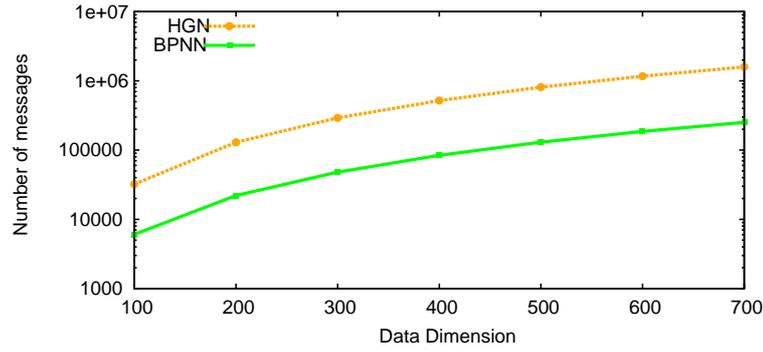


(b) Graph 2: DHGN and P2P-GN

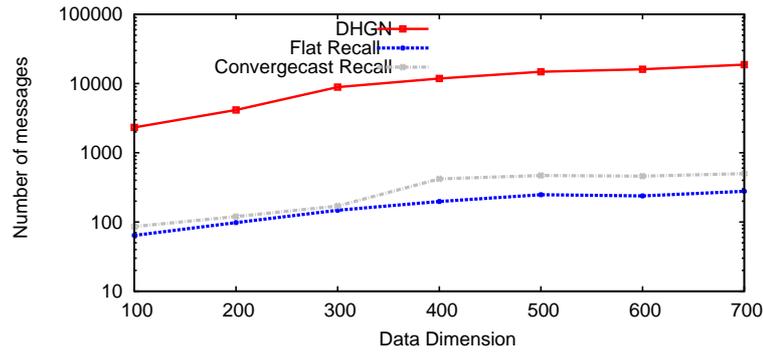
Figure 4.7: Number of messages with increasing data dimension during learning.

Unlike the P2P-GN, DHGN and HGN which perform single-cycle online learning, BPNN requires several iterations. The number of iterations is usually large (thousands) before convergence into an optimal model, particularly for high-dimensional datasets [134]. Considering that the number of learning iterations in this analysis is set to a very small value (ten iterations), the large number of messages in the BPNN demonstrates its inefficiency for high-dimensional problems.

The communication cost during recall in the HGN is the same as during learning (see Figure 4.8). The BPNN consumes less communication during recall than its learning phase as the recall phase does not involve any iterative process. The communication overhead during recall in the P2P-GN depends on the choice of recall approach—the number of messages during recall using the flat recall is smaller than by using the convergecast recall. In a flat recall, a requester sends out n_H recall requests and receives n_H responses and this doubles the communication overhead compared with those during learning. The tree structure aggregation process in the



(a) Graph 1: BPNN and HGN



(b) Graph 2: DHGN and P2P-GN (flat recall and convergecast recall)

Figure 4.8: Number of messages with increasing data dimension during recall.

convergecast recall further increases the communication overhead. The trade-off, however, comes with higher scalability for the convergecast recall when the data dimension (d) is large since every node in the convergecast recall communicates with a constant number of nodes, while, every peer in the flat recall communicates with at most d nodes. However, both of these recall approaches and the DHGN recall incur less communication overhead than that of the BPNN and HGN.

A base station in the HGN requires it to communicate with all the active nodes in the HGN, while, a node in the BPNN requires several message exchanges since the BPNN requires several iterations. This explains that the P2P-GN and DHGN are more communication-efficient than the BPNN and HGN. Among these four methods, the P2P-GN is the most communication-efficient during learning and recall. Its communication complexity is low with respect to the data dimension; this reflects its scalability in dealing with high-dimensional problems.

In general, BPNN incurs the highest number of messages compared to the HGN, P2P-GN and DHGN during learning. Nonetheless, it incurs lower number of messages during recall compared to the HGN. This is because it only involves a single iteration of the feed-forward phase during recall. Among the three GN-based methods (HGN, P2P-GN and DHGN), the P2P-GN requires the smallest number of messages and HGN requires the largest number of messages during both, learning and recall.

4.3 Evaluation Resource Efficiency for Large Datasets

If the computational overheads per peer can be kept low despite the increasing number of training data and data dimension, then the distributed classification algorithm is considered as scalable for large datasets. In this section, we aim to show that the P2P-GN is time-efficient and space-efficient. We prove this through a series of experiments, as follows.

- Experiment 1: Evaluating the efficiency on two large datasets (with a million samples of each dataset): generated using the Waveform (Version 2) and the LED data generators. In this experiment, we demonstrate the storage efficiency and time efficiency of the P2P-GN. This includes the space efficiency of the P2P-GN against increasing training dataset size.
- Experiment 2: Evaluating the time efficiency of the P2P-GN against increasing training dataset size and data dimension.

All experiments were run on a PC with Intel Core i-7 2.9GHz and 8Gb RAM and they were implemented using Java. The distributed algorithm was deployed on a Peersim simulator [136] to simulate the P2P environment specifically the Chord overlay network which consisted of 5,000 peers. We randomly selected a node to act as a requester node which issued the learning and recall queries throughout these experiments. These experimental settings were default throughout the experiments, unless mentioned otherwise.

4.3.1 The Datasets

We used the Waveform data generator (Version 2) and LED data generator because they are available publicly in the UCI repository [13]—to ensure the experiments are reproducible and because of the high number of instances that can be generated. These data generators were used to create two large datasets (Waveform-40 by using Waveform data generator (Version 2) and LED dataset by using LED data generator) with one million instances. In addition, we used the MNIST dataset (which is also available from the UCI repository) for the experiments involving high-dimensional problems. A brief explanation on the MNIST, Waveform-40 and LED datasets are given as follows.

MNIST. As previously described in Section 3.2.2 of Chapter 3, the MNIST was selected because it is one of the largest datasets for classification problems from the UCI database. It represents a handwritten classification problem; its task is to classify a handwritten sample into a correct digit from 10 digits $\{0,1,2,3,4,5,6,7,8,9\}$. It consists of 70,000 samples with 784 number of attributes. The outcome of the experiments (involving the MNIST dataset) suggest the potential of the proposed algorithm for a large-scale distributed handwritten recognition within P2P networks. This may benefit the P2P applications to provide their users with a friendlier user-interface.

LED. The generated LED dataset has one million samples; it was created by a data generator which was originally used in Breiman et al. [26]. The task is to classify a LED display into ten concepts by using 17 attributes. The first seven attributes represent seven light-emitting diodes, while, the next ten attributes are noise attributes. All of these attributes have 10% chance to be inverted in every generated sample.

Waveform-40. The Waveform data generator (Version 2) was originally available from Breiman et al. [25]. The generator created the Waveform-40 dataset which has

one million samples; this provides us with a sufficiently large number of samples to explore the scalability of our scheme for large datasets. The dataset has three classes of waves and 40 attributes which includes 19 noise attributes with mean 0 and variance 1.

4.3.2 Experiment 1: Evaluating Resource Efficiency on the Waveform-40 and LED datasets

The objective of this experiment is to show the high computational overheads at a single host in the centralised implementations of state-of-the-art algorithms (BPNN, radial basis function network (RBFN), nearest neighbour (1-NN), k -nearest neighbour (k -NN) and ID3). We aim to show the effectiveness and efficiency of our algorithm to maintain low computational overheads per host through a distributed approach. In addition, this demonstrates its scalability for large datasets. We used the Waveform-40 and LED datasets in this experiment.

The parameter k in the k -NN experiment is set to three, while, the number of clusters in the RBFN experiment is two. The BPNN in this experiment has one hidden layer with the number of hidden neurons is $\lceil \frac{d+|C|}{2} \rceil$ where d is the data dimension (inputs) and $|C|$ the number of classes (outputs). All of the state-of-art algorithms were implemented using Weka package [75] on a centralised setting.

In the P2P-GN experiment, the selected parameters for both datasets are as follows: $d_s = 3$, $OV = 2$ and $m = 10$. Note that d_s and OV are required for the flat recall, while, all of the three parameters (d_s , OV and m) are required for the convergecast recall. These parameter values were found to be the best after validation using manual tuning. The maximum connections per peer (m) is set to 10 here as this provides a realistic value although it can be considered low in a real P2P system (Table C.1 of Appendix C provides information on the selection of m based on a real world P2P application.).

For the LED problem, the resulting structure of the P2P-GN consists of 22 leaf nodes. The convergecast recall tree structure for this task has a height of two and it has five internal nodes. While, the resulting structure of the P2P-GN in the Waveform-40 experiment consists of 38 leaf nodes and the convergecast recall tree has 15 internal nodes with height equals to three.

Evaluation Metrics

The experiment involves algorithms on two different kinds of architectures—centralised architecture where an algorithm is implemented on a single host and distributed architecture where an algorithm is distributed on several hosts. Therefore, two different sets of metrics are used to evaluate the computational overheads per host of the algorithms in this experiment: evaluation metrics for centralised algorithms and evaluation metrics for distributed algorithms. The set of evaluation metrics that are used to evaluate centralised algorithms include the average learning computation time, $t[L]_\mu$; the average recall computation time, $t[R]_\mu$; and the storage overhead, S . For the P2P-GN (which is a distributed algorithm), we decided to use two evaluation metrics: the maximum processing time taken per host for an instance, $t[P]_{max}$ and the average storage overhead, S_μ . S and S_μ metrics calculate the storage overhead after storing all training samples and these are measured by using a java agent in the `java.sizeOf` package [159].

S is obtained by measuring the storage size of the centralised classifiers, while, $t[L]_\mu$ and $t[R]_\mu$ are calculated by using Equations 4.2 and 4.3, respectively as follows.

$$t[L]_\mu = \frac{\sum_{\mathbf{x} \in \mathcal{X}_{learn}} t_{l,\mathbf{x}}}{|\mathcal{X}_{learn}|} \quad (4.2)$$

where $t_{l,\mathbf{x}}$ is the training time for an instance \mathbf{x} and \mathcal{X}_{learn} is a set of training instances.

$$t[R]_\mu = \frac{\sum_{\mathbf{x} \in \mathcal{X}_{recall}} t_{r,\mathbf{x}}}{|\mathcal{X}_{recall}|} \quad (4.3)$$

where $t_{r,\mathbf{x}}$ is the recall time for an instance \mathbf{x} and \mathcal{X}_{recall} is a set of test instances. Note that $t[L]_\mu$ for the BPNN, RBFN and ID3 are the average time taken by dividing the training time after training 660,000 samples with the total number of samples. This is because these algorithms are batch learning algorithms unlike the k -NN, 1-NN and P2P-GN which perform online learning.

In evaluating the performance of the P2P-GN, the metrics $t[P]_{max}$ and S_μ are computed as follows. We record the learning time for every training instance and the recall time for every test instance at all hosts. Among these hosts, we select the host with the highest processing time and the processing time is recorded as the highest per host processing time for the learning or recall per instance. $t[P]_{max}$ is formally defined in Equation 4.4 as follows.

$$t[P]_{max} = \arg \max_{p \in G_R} t[p] \quad (4.4)$$

where $t[p]$ is the processing time of an instance at a host $p \in G_R$ and G_R is a set of participating hosts. Here, we provide the mean processing time (mean $t[P]_{max}$) of all instances. In calculating S_μ , we initially record the storage overhead at every host, $S[p]$ for all $p \in G_R$ after storing all training samples. Then, we calculate the average storage overhead across the networks (considering all hosts) as follows (see Equation 4.5).

$$S_\mu = \frac{\sum_{p \in G_R} S[p]}{|G_R|} \quad (4.5)$$

Results

The storage overhead, learning time and recall time for the Waveform-40 and LED dataset were presented in Table 4.2 to show the processing overheads incurred in the centralised algorithms (1-NN, k -NN, BPNN, RBFN and ID3). Then, the processing overheads in the P2P-GN which include average storage overhead and the processing time in the P2P-GN operations (i.e., learning, flat recall and convergecast recall) are presented in Table 4.3. We compare the learning time per host, $t[L]_\mu$ in Table 4.2

with $t[P]_{max}$ (learning) in Table 4.3. This is followed by comparing the recall time per host, $t[L]_{\mu}$ of centralised classifiers in Table 4.3 with $t[P]_{max}$ (flat recall and convergecast recall) in Table 4.3.

Table 4.2: S , $t[L]_{\mu}$ and $t[R]_{\mu}$ of the centralised state-of-the-art algorithms on the Waveform-40 and LED dataset. $t[L]_{\mu}$ is the average learning computation time per instance and $t[R]_{\mu}$ is the average recall computation time per instance; and S is the storage overhead.

Classifiers	Waveform-40			LED		
	S (MB)	$t[L]_{\mu}$ (ms)	$t[R]_{\mu}$ (ms)	S (MB)	$t[L]_{\mu}$ (ms)	$t[R]_{\mu}$ (ms)
BPNN	23.96	0.8631	0.0260	15.90	0.1315	0.0122
RBFN	4.51	1.4269	0.0124	0.1428	1.6625	0.0334
1-NN	239.28	0.0003	202.00	81.73	0.0006	194.68
k -NN	239.28	0.0003	95.02	81.73	0.0006	65.12
ID3	26.27	0.0875	0.0006	110.20	0.1134	0.0010

Table 4.3: The average storage overhead S_{μ} of the P2P-GN and the highest processing time per instance $t[P]_{max}$ in the P2P-GN operations (learning, flat recall and convergecast recall) on the Waveform-40 and LED dataset.

S_{μ} (MB)			
Waveform-40		LED	
0.3456		0.0835	
$t[P]_{max}$ (ms)			
Waveform-40		LED	
Learning	0.3456	Learning	0.0054
Recall:		Recall:	
Flat	0.0701	Flat	0.0354
Convergecast	0.0273	Convergecast	0.0258

As we can see, the learning times of the 1-NN and k -NN are the quickest among all methods. The learning time in the BPNN is high considering that the number of iterations is only ten in this experiment^{4.5}. In contrast, the k -NN and 1-NN took a long recall time while ID3, BPNN and RBFN on the other hand, have a brief recall time. The P2P-GN and ID3, however, use a low processing time per host in both learning and recall processing—the learning time in ID3 is higher than the P2P-GN but, its recall time is lower. We found that the recall time per host is smaller in the convergecast recall than in the flat recall. This is because a requester node in the

^{4.5}BPNN usually requires thousands of iterations for convergence.

flat recall aggregates local results from all leaf nodes, while, a requester or internal nodes in the convergecast recall only aggregates the local results from a constant number of peers.

We compare the storage overhead per host, S in Table 4.2 with S_μ in Table 4.3. The results show that the k -NN and 1-NN, both of which perform online learning use the largest amount of storage since they store all the training examples directly. The average storage overhead per host (S_μ) is 0.3456 MB for the Waveform-40 dataset and 0.0835 MB for the LED dataset. The distributed nature of the P2P-GN, results in significantly less storage per host compared to centralised implementation of BPNN, RBFN, 1-NN, k -NN and ID3 since an increase in the number of peers may decrease the storage overhead.

We further conducted an experiment to demonstrate the storage efficiency of the P2P-GN algorithm. The LED dataset was used here, where the number of training instances was increased gradually by 5,000 from 5,000 to 40,000 and the average size of storage was recorded for every 5,000. As shown in Figure 4.9, the storage complexity for the P2P-GN increases slowly. After storing 40,000 training samples, S_μ is only 0.0138 MB. This reflects the lightweights and scalability of these methods, with respect to the storage overhead for large datasets.

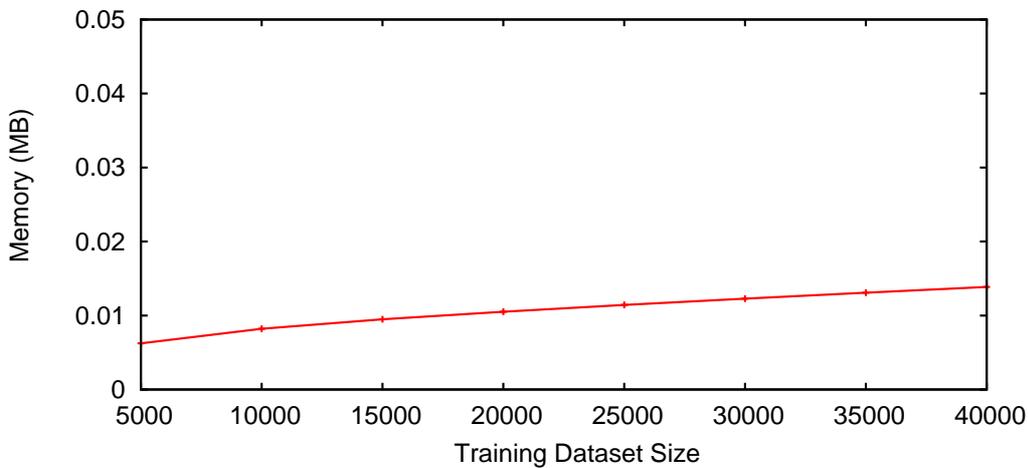


Figure 4.9: Average storage requirement, S_μ in the P2P-GN with increasing training dataset size.

This report highlights the benefit of having a distributed, online learning algorithm—the P2P-GN—within P2P networks which is scalable for large datasets. Although ID3 requires small computational overheads, these methods operate in batch mode and needs all data to be located at a single location; which is not feasible for P2P networks. However, particularly for P2P networks, a distributed tree inducer has been proposed to provide an incremental learning within P2P networks [17]. This, however, builds a tree with a predefined depth which may lower the accuracy compared to the centralised implementation and demands an expensive communication cost.

4.3.3 Experiment 2: Evaluating Time Efficiency on the MNIST dataset

We performed a series of experiments to investigate the time efficiency of our scheme—to demonstrate the scalability of our scheme in dealing with large datasets. In these experiments, we used varying numbers of training instances and data dimension on the MNIST dataset. The experimental set-up and the results are presented as follows.

Experiment Set-up

In this experiment, we implemented the P2P-GN using a cluster-based implementation where a subset of peers within the P2P network are selected to participate in the collaborative classification process. This makes it easier for us to observe the overhead per host in this experiment effectively. The operations in the P2P-GN requires a search function to find a bias element of a specific identifier (where search functions mostly depend on the stored data). Since the number of stored bias elements per host in the network-wide implementation is smaller than the cluster-based implementation when d is smaller than the network size (refer to Section 3.2.6 of Chapter 3), we can assume that the time efficiency per host of the cluster-based

P2P-GN are worse than its network-wide implementation counterpart. With respect to the storage structure, we used hash map in our implementation for its efficiency.

Here, we placed all bias elements for the leaf node ρ_i at a pre-determined peer p_i instead of using DHT to place the bias elements across the network. We also pre-selected the peers to be assigned as internal nodes. Then, an observer was placed at each of these peers to record the local learning time per instance, t_L and the local recall time per instance, t_R for each operation. The learning and recall queries were submitted from random peers within the network.

The experiments explored the following.

- t_L and t_R with an increase in the number of training instances.
- t_L and t_R with an increase in data dimension.

For the experiments involving increasing data dimension, the original MNIST dataset was used. From it, we created seven new datasets. For each set, we selected j -number of features out of 784 from the original feature set and each set consists of 60,000 training instances and 10,000 recall instances. From a table perspective, the elements of dataset were arranged into $n \times j$ where the original MNIST dataset consists of $70,000 \times 784$ elements where the number of columns (number of attributes), j is 784 and the number of rows (dataset size), n is 70,000. Initially, we set j equals to 100 and we selected attributes from the first to j th column from the original dataset. The resulting new dataset has $70,000 \times 100$ elements. This created the first dataset and the same process was repeated with different j value where j was gradually increased by 100 until 700. As a result, seven datasets with seven different data dimensions (i.e., $\{70,000 \times 100, 70,000 \times 200, 70,000 \times 300, 70,000 \times 400, 70,000 \times 500, 70,000 \times 600, 70,000 \times 700\}$) were obtained at the end.

Local Processing Time with Training Dataset Size

We randomly selected six peers which hosting leaf nodes and placed an agent as observer at each of these peers to watch their local learning time. The observer

recorded the local learning time for every instance and in total, 10,000 training instances from MNIST dataset (in their original 784 dimension) were used in this experiment. The learning t_L at the observed peers are reported in Figure 4.10; all

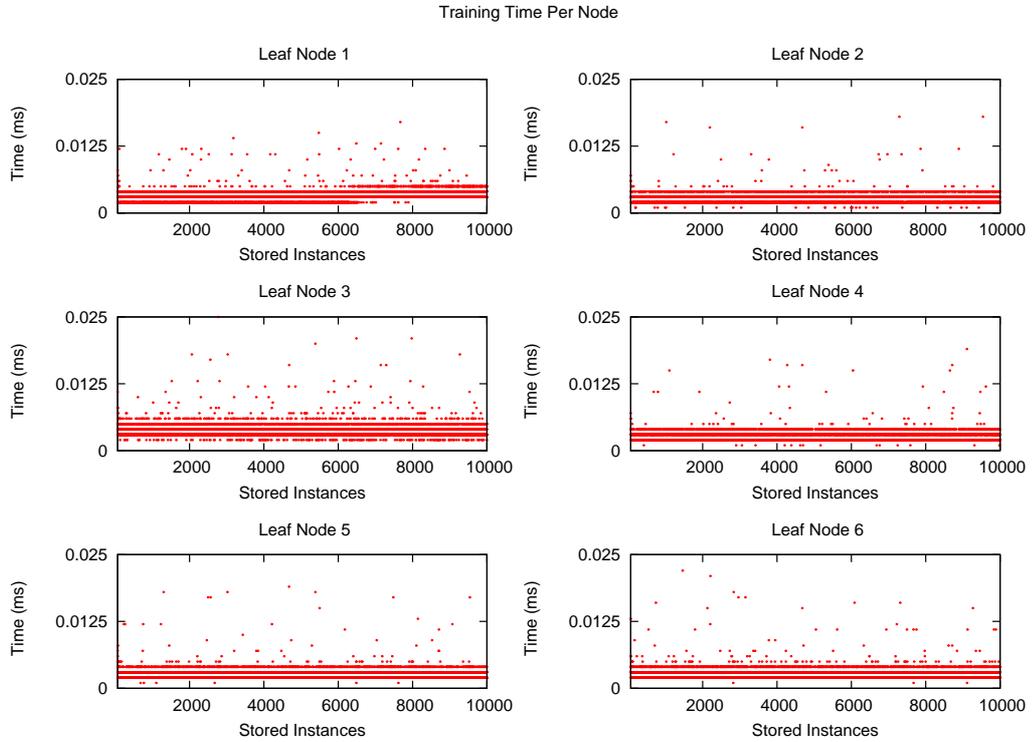


Figure 4.10: Six peers which hosting leaf nodes by local learning time per instance (t_L) in the experiment on MNIST dataset.

of them exhibit constant learning time with increasing number of stored instances. This satisfies our aim to ensure the low computation overhead per node in this scheme. Furthermore, the results also demonstrate the ability of our algorithm to perform online learning.

We further demonstrate the time efficiency of our scheme during recall. In this experiment, seven simulation runs are conducted with vary number of training instances. Starting from 5,000 training instances in the first run, the number of training instances was increased gradually in the order of 5,000 to 35,000. We observed the recall time for 10,000 recall observations during each run. We recorded the local recall time of six random hosts. From these six hosts, three are leaf nodes, two are internal nodes and one is a requester node in the convergecast P2P-GN.

While in the P2P-GN, only four nodes were observed, where out of the four hosts, one is a requester and the rest are leaf nodes. The results are provided in Table 4.4 and Table 4.5.

Table 4.4: Local recall time, t_R (ms) in the flat recall with increasing stored data.

Host		Training Dataset Size						
		5,000	10,000	15,000	20,000	25,000	30,000	35,000
Leaf node 1	mean	0.00115	0.00113	0.00124	0.00126	0.00119	0.00108	0.00122
	sd	0.00071	0.00068	0.00069	0.00079	0.00105	0.00071	0.00126
Leaf node 2	mean	0.00115	0.00119	0.00124	0.00131	0.00129	0.00123	0.00128
	sd	0.00077	0.00068	0.00084	0.00083	0.00095	0.00072	0.00069
Leaf node 3	mean	0.00120	0.00123	0.00128	0.00116	0.00126	0.00128	0.00121
	sd	0.00069	0.00068	0.00085	0.00073	0.00141	0.00100	0.00064
Requester	mean	0.15797	0.17245	0.18160	0.19884	0.20297	0.20656	0.21077
	sd	0.03204	0.03799	0.04131	0.04499	0.04787	0.04307	0.04523

Table 4.5: Local recall time, t_R (ms) in the convergecast recall with increasing stored data.

Host		Training Dataset Size						
		5,000	10,000	15,000	20,000	25,000	30,000	35,000
Leaf node 1	mean	0.00202	0.00176	0.00191	0.00184	0.00225	0.00204	0.00194
	sd	0.00087	0.00117	0.00089	0.00082	0.00117	0.00088	0.00078
Leaf node 2	mean	0.00196	0.00189	0.00189	0.00191	0.00239	0.00200	0.00197
	sd	0.00080	0.00084	0.00078	0.00086	0.00153	0.00115	0.00108
Leaf node 3	mean	0.00206	0.00205	0.00193	0.00190	0.00234	0.00199	0.00193
	sd	0.00095	0.00083	0.00087	0.00087	0.00115	0.00101	0.00076
Internal node 1	mean	0.00755	0.00734	0.00759	0.00753	0.00757	0.00753	0.00777
	sd	0.00215	0.00214	0.00233	0.00239	0.00237	0.00224	0.00220
Internal node 2	mean	0.00687	0.00673	0.00696	0.00719	0.00703	0.00697	0.00701
	sd	0.00186	0.00208	0.00234	0.00220	0.00226	0.00248	0.00187
Requester	mean	0.06039	0.05956	0.06109	0.06142	0.07549	0.06122	0.06449
	sd	0.01228	0.01099	0.01163	0.01114	0.00889	0.01245	0.01248

As shown in Table 4.4, t_R at the requester in the flat recall increases very slowly with an increase in training dataset size in this experiment, when we would otherwise expect to see a constant growth at every node. Let \tilde{s} be the number of nodes whose outputs are combined. The increase in training examples can increase the conflicts in determining the best decision when there are many local classifiers (\tilde{s}) involved during the function `finalPrediction`. The node which calculates the final decision may receive many conflicted predictions from different local classifiers and hence it is difficult to decide the best predictions. As can be seen in Table 4.5, the computational time at the requester using convergecast recall remains small as

\tilde{s} is small (since \tilde{s} is capped by m). Therefore, given q as the number of training data, there is a function of q that represents the conflicts as the difficulties of these conflicts grow, with an increase in q , and this function is associated with \tilde{s} . However, the reports show that the effect of q here is extremely small.

Local Processing Time with Data Dimension

We now study the complexity of training time per node for different data dimensions of the P2P-GN based on 10,000 observations. For simplicity, we use the following parameters: $d_s = \lceil \log_2 d \rceil$, $OV = \lceil d_s 2 \rceil$ and $m = 10$. By doing so, we can limit the segment size to be sufficiently small relative to d . The system learnt 10,000 training instances where an instance was submitted to the system for every 5 seconds from random peers.

Table 4.6 shows that with an increase in data dimension, the learning t_L is constant for every leaf nodes. Six hosts from the peers which were assigned with leaf nodes are randomly selected. An observer agent was located at every selected peer to record the local learning time per instances. Since the peers only required to perform a simple pattern matching operation to match each sub-pattern to the stored sub-pattern at the peers' storage, the local learning time is very fast. In addition, the use of hash map in our implementation also contributes for fast searching and storing which are two of the main procedures in the P2P-GN learning.

Table 4.6: Local training time, t_L (ms) with increasing data dimension.

Host		Data Dimension (d)					
		100	200	300	400	500	600
Leaf node 1	mean	0.00377	0.00392	0.00395	0.00380	0.00266	0.00240
	sd	0.00100	0.00116	0.00096	0.00098	0.00076	0.00069
Leaf node 2	mean	0.00249	0.00275	0.00266	0.00263	0.00284	0.00360
	sd	0.00071	0.00083	0.00077	0.00077	0.00079	0.00100
Leaf node 3	mean	0.00247	0.00257	0.00265	0.00260	0.00413	0.00245
	sd	0.00075	0.00079	0.00075	0.00077	0.00104	0.00072
Leaf node 4	mean	0.00263	0.00283	0.00264	0.00274	0.00307	0.00310
	sd	0.00085	0.00107	0.00138	0.00117	0.00103	0.00112
Leaf node 5	mean	0.00274	0.00317	0.00283	0.00296	0.00274	0.00264
	sd	0.00082	0.00096	0.00087	0.00085	0.00084	0.00075
Leaf node 6	mean	0.00228	0.00251	0.00268	0.00258	0.00294	0.00265
	sd	0.00072	0.00074	0.00077	0.00075	0.00092	0.00079

Table 4.7: Local recall time, t_R (ms) during flat recall with increasing data dimension.

Host		Data Dimension (d)					
		100	200	300	400	500	600
Leaf node 1	mean	0.00150	0.00137	0.00155	0.00155	0.00140	0.00133
	sd	0.00112	0.00107	0.00086	0.00092	0.00081	0.00088
Leaf node 2	mean	0.00148	0.00164	0.00143	0.00135	0.00139	0.00137
	sd	0.00111	0.00095	0.00087	0.00091	0.00090	0.00090
Leaf node 3	mean	0.00151	0.00144	0.00145	0.00144	0.00147	0.00137
	sd	0.00112	0.00087	0.00086	0.00091	0.00103	0.00094
Leaf node 4	mean	0.00156	0.00143	0.00143	0.00137	0.00140	0.00199
	sd	0.00122	0.00095	0.00089	0.00088	0.00087	0.00098
Leaf node 5	mean	0.00143	0.00160	0.00140	0.00148	0.00140	0.00143
	sd	0.00111	0.00093	0.00089	0.00085	0.00086	0.00087
Requester	mean	0.05258	0.05975	0.08017	0.10282	0.12680	0.12139
	sd	0.02287	0.02687	0.03446	0.03826	0.04331	0.03992

Table 4.8: Local recall time, t_R (ms) during convergecast recall with increasing data dimension.

Host		Data Dimension (d)					
		100	200	300	400	500	600
Leaf node 1	mean	0.00229	0.00211	0.00186	0.00225	0.00216	0.00205
	sd	0.00122	0.00119	0.00110	0.00126	0.00098	0.00103
Leaf node 2	mean	0.00220	0.00220	0.00194	0.00213	0.00216	0.00197
	sd	0.00123	0.00116	0.00105	0.00109	0.00110	0.00107
Leaf node 3	mean	0.00218	0.00225	0.00188	0.00210	0.00222	0.00194
	sd	0.00116	0.00115	0.00107	0.00102	0.00121	0.00108
Internal node 1	mean	0.00943	0.01064	0.00969	0.00996	0.00614	0.00600
	sd	0.00492	0.00482	0.00463	0.00389	0.00264	0.00245
Internal node 2	mean	0.00966	0.01155	0.00890	0.00918	0.00683	0.00589
	sd	0.00491	0.00481	0.00445	0.00395	0.00290	0.00251
Requester	mean	0.05178	0.05954	0.06215	0.06065	0.07036	0.06023
	sd	0.04490	0.04047	0.04465	0.03877	0.03686	0.03657

Next, we report the local recall time, t_R at six hosts based on 10,000 observations. In the flat recall experiments, we randomly selected five peers among the peers which hold the leaf nodes and a requester. While in the convergecast recall experiments, we chose three peers among the peers which were assigned with leaf nodes, two peers among the peers which were assigned as internal nodes, and a requester peer. An observer agent was placed at every selected peer to watch the local recall time. The results are provided in Table 4.7 and Table 4.8.

t_R at all of the observed leaf nodes in the flat recall and the convergecast recall are constant with an increase in data dimension. While at the requester node in the flat recall, t_L increases linearly. The learning time at the requester in the convergecast approach, however, is independent of the data dimension, d since we have limited connection per peer to a constant m . In the results of the flat recall and the convergecast recall, we found that the mean local recall time at the requester when $d = 500$ is larger than the mean local recall time when $d = 600$ since at $d = 600$, the P2P-GN structure has a larger number of leaf nodes. This is because of the selected parameter d_s and OV (see Table C.4 of Appendix C). However, it is possible to improve this by selecting better parameters which process may require optimisations or cross-validations^{4,6}.

The mean local recall time at the requesters in both the flat recall and the convergecast recall are approximately within the same range (0.05 – 0.06 ms) when the data dimension is 100 and 200. However, when $d = 300$, we can see the obvious difference of local recall time between the two recall approaches. From mean 0.08 ms at $d = 300$, the local recall time at the requester in the flat recall increases to mean 0.12 ms at $d = 600$. The local recall time at the requester in the convergecast recall, however, grows at a constant rate in between a mean of 0.05 ms to 0.07 ms.

^{4,6}Although we do not address this issue here, we have provided some insights on parameter selections in Section 5.4.11 of Chapter 5

4.4 Complexity Analysis

This section theoretically evaluates the P2P-GN against the storage complexity, local computational complexity at the peer level and overall computational complexity at the network level. We use the Big O notation to describe the computational complexity here. The list of notations that are used in this section is available in Table A.2 of Appendix A. Let d be the data dimension, d_s be the sub-pattern size, u be the size of input domain and q be the number of stored instances. The number of sub-patterns or leaf nodes is denoted by n_H where n_H is $\mathcal{O}(d)$. In this analysis, the storage overhead is firstly estimated; followed by the computational complexity at the peer level and the overall time complexity at the network level.

4.4.1 Storage Complexity Estimation

Let u^{d_s} be the number of possible combinations of distinct sub-patterns with length d_s which can be stored at a each leaf node and u^d be the number of possible combinations of distinct patterns with length d which can be stored in the whole system where $d > d_s$. Generally, the P2P-GN has a high memory capacity since it can store at most u^d distinct patterns. This shows its memory capacity increases exponentially with an increase in d and linearly with an increase in u . The estimated number of bias elements is n_b for a leaf node is given in Equation 4.6.

$$\begin{aligned}
 n_b &\approx \frac{u^{d_s}}{u^d} \cdot q \\
 &\approx q \cdot u^{(d_s-d)} \\
 &\approx \frac{q}{u^{-(d_s-d)}} \\
 &\approx \frac{q}{u^d}
 \end{aligned} \tag{4.6}$$

then we denote $\tilde{d} = -(d_s - d) = d - d_s$ where $d_s - d < 0$ since $d > d_s$.

Assume that the minimum value of d_s is $d_{s,min} = 2$, the maximum value of d_s is $d_{s,max} = d - 1$. Let $\tilde{d}_{min} = d - d_{s,min}$ and $\tilde{d}_{max} = d - d_{s,max}$ where $d_{s,max} > d_{s,min}$ and

$d_{s,max}, d_{s,min} < d$. Then, $\tilde{d}_{max} < \tilde{d}_{min}$ and so $u^{\tilde{d}_{max}} < u^{\tilde{d}_{min}}$. Hence, n_b is decreasing as d_s is approaching $d_{s,min}$, while, it is increasing as d_s is approaching $d_{s,max}$.

The total number of bias elements within the system (including all leaf nodes) can be estimated as n_b times the number of leaf nodes (n_H). We obtained the estimate of total number of bias elements, $n_{b,total}$ which is given in Equation 4.7.

$$n_{b,total} \approx n_b \cdot n_H \quad (4.7)$$

Considering that the bias elements are uniformly distributed among N peers, then the estimated number of bias elements per peer, $n_{b,total}/P$ is given in Equation 4.8 as follows.

$$n_{b,total}/P \approx \frac{n_{b,total}}{N} \quad (4.8)$$

where $N > 0$.

In conclusion, an increase of stored data increases n_b , however, a decrease in sub-pattern size reduces it quadratically and an increase in the number of input domain reduces it in a linearly (as shown in Equation 4.6). The decrease in sub-pattern size reduces n_b and increases n_H , while, the growth of n_H or n_b effectively results in the growth in the total storage overhead. Nonetheless, an increase in network size (N) reduces the storage overhead per peer (as shown in Equation 4.8).

4.4.2 Local Computational Complexity At Peer Level

The data structure plays an important role to determine the execution time complexity of this scheme. The learning and recall performance in the P2P-GN depends on the searching, as a basis operation. The analysis in this section is made based on assumption that a hash table is used as the data structure for storage in this scheme. The average performance of hash tables are shown in Table 4.9. Then, the summary on the local computational complexity for training and recall per instance are given in Table 4.10 and Table 4.11, respectively.

Table 4.9: Hash table average performance

Operation	Complexity
insert	$\mathcal{O}(1)$
find	$\mathcal{O}(1)$
delete	$\mathcal{O}(1)$

Local Learning Time The learning time at leaf node, $t_{localLearn}$ ^{4.7} depends on the searching and insertion operation performance as it is the main procedure in the local learning in our scheme. Given that the hash table is used for storage, then the learning complexity per node is constant.

Local Recall Time The local recall time complexities at leaf nodes, $t_{localRecall}$ ^{4.8} are independent of an increase in data dimension, d and an increase in dataset size, q . The processing time during aggregation, t_{agg} depends on \hat{r} where \hat{r} is the number of inputs. Hence, the time complexity at a requester in the flat approach is linear as a function of d . The experiments in this chapter have shown that the recall time at a requester in the flat approach is slightly affected by $g(q)$ —that is a magnitude of conflicts which is a function of q . In contrary, the requester’s recall time in the convergecast approach is independent of d since the aggregation only involves a constant m peers instead of d and $g(q)$ does not effect the recall time in the requester. This shows that the recall time complexity per node in all nodes

Table 4.10: Local computational complexity (Big O) of the P2P-GN as a function of training dataset size (q) at the peer level

Time Complexity			
Operation	Leaf nodes	Requester	Internal nodes
Learning	constant	constant	*
Flat recall	constant	$\mathcal{O}(g(q))$	*
Convergecast recall	constant	constant	constant
Communication Complexity			
Operation	Leaf nodes	Requester	Internal nodes
Learning	constant	constant	*
Flat recall	constant	constant	*
Convergecast recall	constant	constant	constant
* refers to nodes which do not involve in the operation			

^{4.7}The execution time of function `localLearning(.)`.^{4.8}The execution time of function `localRecall(.)`.

Table 4.11: Local computational complexity (Big O) of the P2P-GN as a function of data dimension (d) at the peer level

Time Complexity			
Operation	Leaf nodes	Requester	Internal nodes
Learning	constant	linear	*
Flat recall	constant	linear	*
Convergecast recall	constant	constant	constant
Communication Complexity			
Operation	Leaf nodes	Requester	Internal nodes
Learning	constant	linear	*
Flat recall	constant	linear	*
Convergecast recall	constant	constant	constant
* refers to nodes which do not involve in the operation			

except the requester nodes is independent of the growth in dataset size. Besides that, the computation complexity at the requester node was influenced by q on a very small scale. The recall time at the requester in the flat strategy grows linearly with an increase in data dimension, while, the recall time at the requester in the convergecast strategy grows logarithmically. These reflect the resource efficiency and scalability of the P2P-GN in dealing with large and high-dimensional dataset.

Local Communication Complexity The communication overhead at a requester peer is $\mathcal{O}(d)$ during learning. The communication overhead at a requester peer in the flat recall is $\mathcal{O}(d)$. If $d > m$, then the communication overhead decreases in the convergecast recall to a constant m . However, the communication overhead at a leaf node remains constant in learning procedure and recall procedure.

4.4.3 Overall Time Complexity At Network Level

During learning, n_H LEARN_REQUEST messages are submitted simultaneously to n_H leaf nodes. The overall learning time $t_O[L]$ is

$$t_O[L] = t_{LQ} + t_{localLearn} \quad (4.9)$$

where t_{LQ} is the communication time to send `LEARN_REQUEST` message. In the P2P-GN, n_H `RECALL_REQUEST` messages are sent to the leaf nodes in parallel. Upon receiving the responses to the queries, the aggregation then taken place at the requester node. The overall recall time in the flat recall $t_O[R_{flat}]$ is given in Equation 4.10 below.

$$t_O[R_{flat}] = t_{RQ} + t_{localRecall} + t_{RR} + t_{agg} + t_{fP} \quad (4.10)$$

where t_{agg} is the execution time for function `agg(.)` at an internal node or a requester; t_{fP} is the execution time for function `finalPrediction(.)` at a requester; t_{RQ} is the communication time to send a `RECALL_REQUEST` message; and t_{RR} is the communication time to send a `RECALL_RESPONSE` message. In the convergecast strategy, the predictions are aggregated iteratively until converge at the root node. Let $t_{agg}[i]$ be the local aggregation at level i . The overall recall time in the convergecast strategy $t_O[R_{convergecast}]$ is formulated in Equation 4.11 as follows.

$$t_O[R_{convergecast}] = \sum_{i=1}^h t_{FQ}[i] + t_{RQ} + t_{localRecall} + t_{RR} + \sum_{i=1}^h (t_{FR}[i] + t_{agg}[i]) + t_{agg} + t_{fP} \quad (4.11)$$

where $t_{FQ}[i]$ is the communication time to send `FORWARD_REQUEST` message at level i , $t_{FR}[i]$ is the communication time to send `FORWARD_RESPONSE` message at level i , $t_{agg}[i]$ is the execution time for function `agg(.)` at an internal node or a requester at level i .

To simplify the equations, we assume that t_{FQ} , t_{FR} and t_{RR} are equal and we denote them by t_1 where t_1 represents a unit communication delay. Then, we can assume that t_{RQ} or t_{LQ} which involves lookup time equal t_s . We further assume that $t_{localRecall}$, $t_{localLearn}$ and t_{fP} equal t_2 . The local aggregation time in the flat recall is $t_{agg} = n_H \times t_2$ for n_H inputs ($\tilde{s} = n_H$) while, in the convergecast recall, $t_{agg} = m \times t_2$ for m inputs ($\tilde{s} = m$) where m is a constant. By substituting t_1 , t_s and t_2 into Equations 4.9, 4.10 and 4.11, then we can estimate $t_O[L]$, $t_O[R_{flat}]$ and $t_O[R_{convergecast}]$ as follows.

$$t_O[L] = t_s + t_2 \quad (4.12)$$

$$t_O[R_{flat}] = t_1 + t_s + t_2(n_H + 2) \quad (4.13)$$

$$\begin{aligned} t_O[R_{convergecast}] &= t_1(2\bar{h} + 1) + t_s + t_{localRecall} + t_{agg}(\bar{h} + 1) + t_{fP} \\ &= t_1(2\bar{h} + 1) + t_s + 2t_2 + t_{agg}(\bar{h} + 1) \\ &= t_1(2\bar{h} + 1) + t_s + 2t_2 + (m \cdot t_2)(\bar{h} + 1) \\ &= t_1(2\bar{h} + 1) + t_s + \bar{h} \cdot m + \bar{h} \cdot t_2 + 3t_2 + m \\ &= t_1(2\bar{h} + 1) + t_s + t_2(\bar{h} + 3) + m(\bar{h} + 1) \end{aligned} \quad (4.14)$$

In the implementation within a Chord network, it can be concluded that the overall learning time in the P2P-GN is $\mathcal{O}(\log N)$. Since n_H is $\mathcal{O}(d)$, then the flat recall time is $\mathcal{O}(d + \log N)$. Given that $\bar{h} = \mathcal{O}(\log_m n_H)$, then $\bar{h} = \mathcal{O}(\log_m d)$. The convergecast recall time is $\mathcal{O}(m \log d + \log N)$.

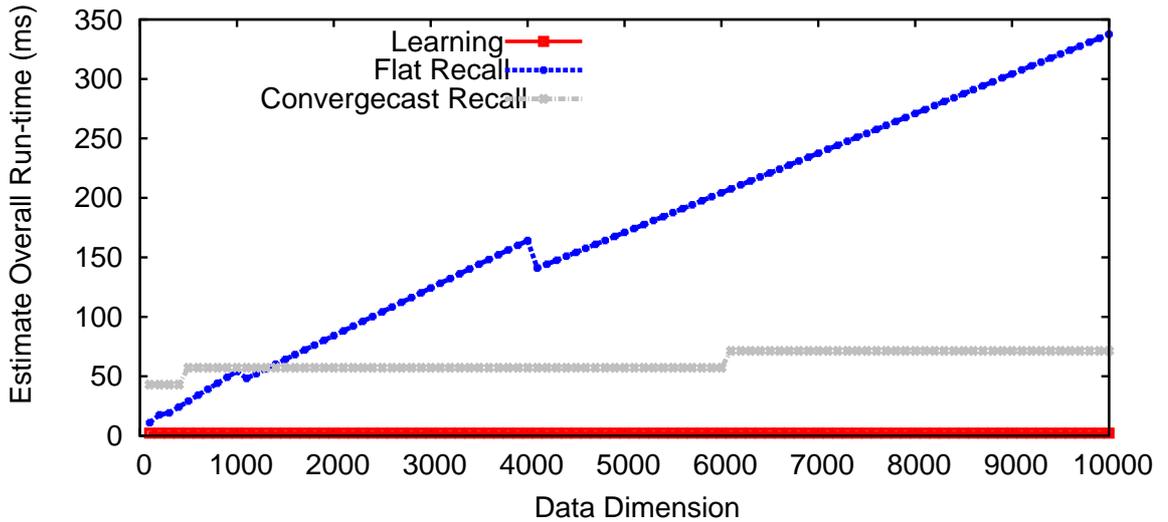


Figure 4.11: Estimated overall run-time with an increase in data dimension.

Under the optimal connection assumption, the overall learning time is constant, while, the flat recall time is $\mathcal{O}(d)$ and the convergecast recall time is $\mathcal{O}(m \log d)$. If m is a very small constant compared to d , the convergecast recall time is $\mathcal{O}(\log d)$. This shows that the convergecast recall improves the efficiency and scalability of the flat recall when $d \gg \bar{h}$ and $d \gg m$. The lookup time t_s is 1 under the assumption of optimal connections^{4.9} and the lookup time is $\mathcal{O}(\log N)$ in the Chord network.

^{4.9}Every peer within the network knows the location of each others so that direct connections among them can be established.

We plot the estimated overall recall time with increasing values of d in Figure 4.11 by using Equations 4.12, 4.13 and 4.14, based on the following assumptions: optimal connections are used with $t_2 = 0.2$ ms, every hop has 2 ms delay ($t_s, t_1 = 2$), $m = 10$, $d_s = \lceil \log d \rceil$ and $OV = \lceil \frac{d_s}{2} \rceil$. The estimated overall run-time of the flat recall is lower than the convergecast recall time when $d \leq 1,300$. This shows that the convergecast recall manages to decrease the overall recall run-time for high-dimensional datasets.

4.5 P2P-GN within Dynamic Networks

P2P network has a logical topology which is dynamic (where peers may join and leave the network any time). As a consequence, the P2P-GN has to deal with the risk of losing bias elements which may degrades the classification accuracy and high latency due to outdated routing information. The improvement on resilience that is achieved by designing a fully-distributed system is insufficient in dealing with a highly dynamic network for a long-term operation, as all bias elements may be lost together with the failed peers over time. This requires more efforts to ensure the availability of the system and minimise the risk of losing bias elements.

Other than erasure coding, replication is the most popular method in providing fault-tolerance within distributed systems [156, 203]. It can be argues that distributed systems, specifically P2P systems have high potential in providing reliability and a high data availability through storing backup data at multiple different physical locations. Compared to a centralised system which is prone to a single point of failure, replication is feasible in any P2P-based application to improve the application reliability, considering the large-scale nature of the networks. Erasure coding is useful to provide reliability for large objects; therefore, it is not considered here since the bias elements are fine-grained and relatively small objects [203]. For these reasons, we chose to apply the replication mechanism in our approach to further improve the reliability of the P2P-GN in dealing with dynamic networks by maintaining the availability of the bias elements. The availability of bias elements which is the key of the P2P-GN reliability is formally defined in Definition 4.4.

Definition 4.4. *Bias element availability at time t , $B_A[t]$ is equal to the number of available bias elements at time t divided by the number of bias elements in a stable network.*

Replication approach is used in the structured P2P networks to maintain their network stability by replicating the routing information [179, 160]. However, these P2P networks do not provide such a replication service to support applications built on top of its overlay and leave this responsibility to the application. Therefore, to ensure the continuous availability of the P2P-GN service, an application-specific fault tolerant mechanism which supports the P2P-GN operation is required.

Replication strategies in structured P2P networks are different in many ways—the placement of the replicas, the degree of replication, and the replica consistency maintenance for mutable contents. Several existing replication schemes include replication using multiple hashing function [211], successor list [95] and symmetric replication scheme [69]. In this thesis, we adopted the successor list strategy, considering its simplicity to maintain the data availability and efficiency to tolerate the leaving peers. Since the location of the replicas in this method are within a small neighbourhood, the peers are aware of changes in topology which require the replicas to update.

Moreover, the peer that holds the original bias element can access the replica copies in a single hop. The static (inmutable) content distribution, such as in the P2P file sharing system, is easier to maintain compared to the mutable content such as in the P2P database systems and in the P2P-GN. This is because an extra effort is needed to ensure the consistency of replicas—one-copy equivalence [91]. There are two ways to ensure this: asynchronous replica update and synchronous replica update.

In asynchronous replica update, any replica has the read-write permission to be updated independently while in the synchronous replica update, the update to the original copy (master copy) of the replica must be completed first before the update to other replicas can be made. The synchronous option highly preserves the one-copy

equivalence criterion, however, it requires expensive overheads when the changes are frequent and it may also delay the update to other replicas. The asynchronous method provides a fast update but at the cost of inconsistencies between replicas which then needs a reconciliation procedure.

In this thesis, we use a synchronous replica update to avoid expensive periodic reconciliation overheads. The expensive replication overheads due to frequent update can also be avoided since the size of a bias element is relatively small compared to other P2P database objects. This allows an efficient replication process in our approach and enables a rapid update which is crucial for an online algorithm to assimilate the frequent learning. The small size of a bias element also minimises overheads at the peer which has the original bias element.

Next, we analyse the ability of the P2P-GN to produce decision with losing bias elements—that is the reliability of the P2P-GN from the distributed approach without extra reliability effort. The distributed nature of the P2P-GN algorithm provides its a degree of robustness in two ways: First, the bias elements store the predictions for small overlapping sub-patterns from the whole pattern which effectively increases the robustness of the algorithms. Second, the fine-granularity of the bias elements and the uniform distribution of these bias elements across the network decrease the risk of losing bias elements when the network is large (see Proposition 4.5).

The advantage of overlapping patterns for the system robustness is given as follows. As shown in Figure 4.12, given k bias elements are created for the pattern P , the pattern P can still be reconstructed from j bias elements where $j < k$.

Proposition 4.5. *Increasing network size increases the P2P-GN reliability.*

Proof. Equation 4.6 explains the effect of increasing network size on the P2P-GN reliability, where the number of bias elements per peer is decreasing while the network size is increasing. Effectively, the small number of bias elements at peer i results in losing only a small amount of bias elements when peer i fails. \square

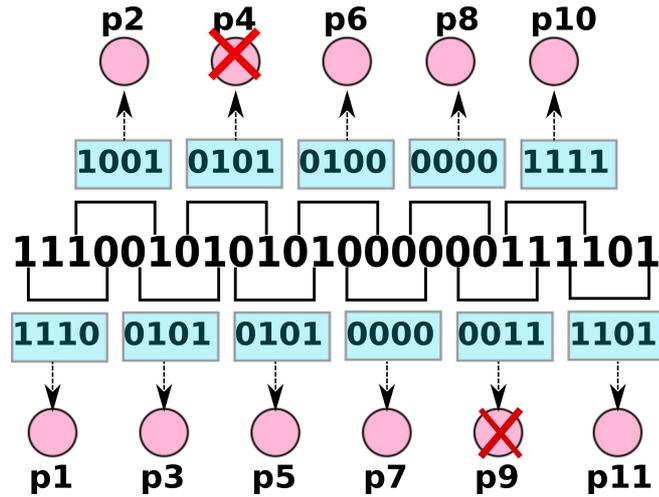


Figure 4.12: An example of the P2P-GN ability to reconstruct the whole pattern in the event of failed peers through overlapping patterns. The sub-patterns in the P2P-GN are located at 11 different peers: p1, p2, p3, p4, p5, p6, p7, p8, p9, p10, and p11. The whole pattern can still be reconstructed although p4 and p9 have failed.

In addition to the available reliability of the P2P-GN, a fault-tolerant scheme is required for the system to operate continuously over the long term within the dynamic networks since the system may lose all the bias elements due to peers leaving over time.

4.6 Fault Management in P2P-GN

This section describes the approach that is used to improve the fault-tolerance of the P2P-GN. Our aim is to improve the bias element availability which effectively improves the P2P-GN reliability. This is achieved through replication which provides high availability and durability of the algorithm in the presence of peers failures and network attacks. Initially, we briefly present the fault management service which is designed as a separate component from the P2P-GN computation and then we show the integration of the information from the underlying P2P network service with the fault management service in Section 4.6.1. This is followed by detail description of protocols that are used upon the joining and departure of peers to ensure the correctness of the P2P-GN in Section 4.6.2 and Section 4.6.3.

4.6.1 Fault Management Framework

The fault management framework to support the P2P-GN operation is described in Figure 4.13. Four services in fault management are neighbourhood watch, replica write, peer selection and replica recovery. The neighbourhood watch service is supported by the stabilisation process which is a protocol in underlying P2P overlay networks; it assists every peer to be aware to the changes of their nearby peers. The replica write service provides a systematic approach for learning phase—this ensures the consistency of replicas. Peer selection is a service to provide a means for the peers to select internal nodes in the convergecast approach. Finally, the replica recovery service includes the recovery processes which are performed to ensure the memories (bias elements) and sufficient replicas of these memories are available across dynamic networks. All of these services work closely with the underlying P2P overlay which provides them with information regarding the network topology and efficient routing.

Neighbourhood Watch Service

In the neighbourhood watch service, every peer keeps track of its neighbourhood. The underlying Chord P2P network overlay stores k -successor list for routing purpose. Hence, our recovery approach uses this information to watch r closest successors and r closest predecessors at every peer, where r is the number of replicas and $r < k$. Therefore, the term successor list here refers to the r successor lists, instead of the k successor list which is used in the overlay network. These set of nodes form a set of neighbours $\Gamma(p)$ of a node p which is r -neighbourhood of p (see Figure 4.14). The formal $\Gamma(p)$ definition is given as follows.

Definition 4.6. *The $\Gamma(p)$ is defined as the r -neighbourhood of a node p which consists of the nodes within the radius r from p where r is the size of the successor list. This includes the closest predecessors and successors, at distance at most r from peer p . The i th predecessor of a peer p is denoted as $p_{pre[i-1]}$ and the i th successor of a peer p is denoted as $p_{suc[i-1]}$.*

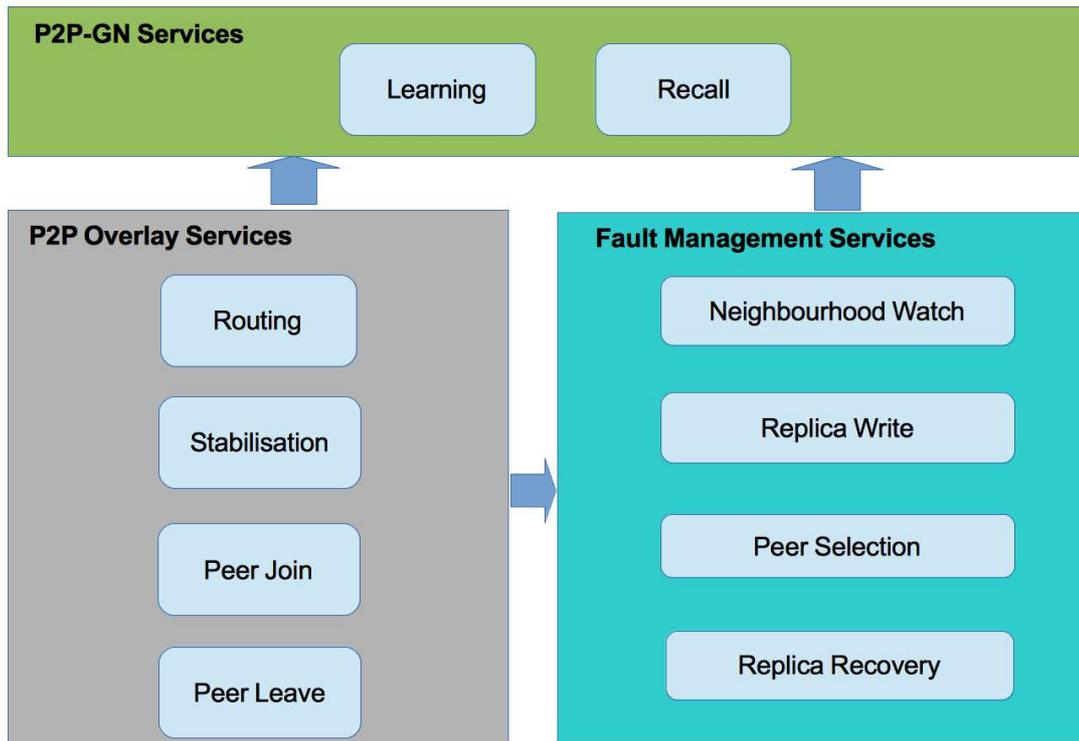


Figure 4.13: The fault management framework of the P2P-GN. The fault management services and the P2P-GN services require the integration with the services from the P2P overlay networks.

The successors and the immediate predecessor of the peer h_{50} in the example in Figure 4.14 are summarised in Table 4.12. As shown in Figure 4.15, the neighbourhood change information is obtained from the **stabilisation** service from the overlay services. The **stabilisation** is a periodical process of an overlay P2P network to correct its routing information.

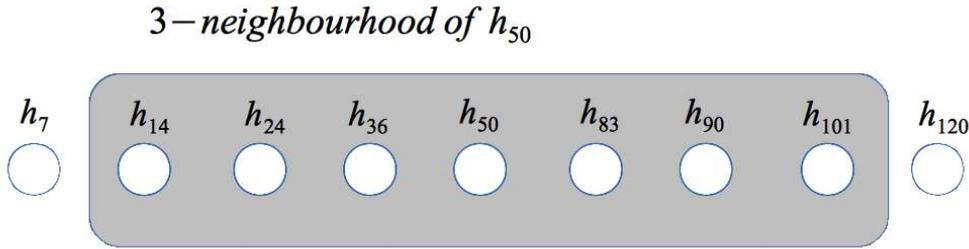


Figure 4.14: An example of a neighbourhood for a peer with identifier h_{50} . Given r -successor list and r -predecessor list where $r = 3$, the r -neighbourhood of the peer h_{50} consists of a group of peers $\{h_{14}, h_{24}, h_{36}, h_{50}, h_{83}, h_{90}, h_{101}\}$.

Table 4.12: Neighbourhood of peer h_{50} from the example in Figure 4.14. $p_{pre}[i]$ is an $(i + 1)$ -th predecessor of peer p where $p_{pre}[0]$ refers to its immediate predecessor. While, $p_{suc}[i]$ is an $(i + 1)$ -th successor of peer p where $p_{suc}[0]$ refers to its immediate successor.

Neighbourhood of peer $p = h_{50}$			
Predecessors		Successors	
Notation	Peer Identifier	Notation	Peer Identifier
$p_{pre}[0]$	h_{36}	$p_{suc}[0]$	h_{83}
$p_{pre}[1]$	h_{24}	$p_{suc}[1]$	h_{90}
$p_{pre}[2]$	h_{14}	$p_{suc}[2]$	h_{101}

Replica Write Service

The replica write service involves updates to the bias elements during learning. Given r is the replication rate, every node keeps the duplicate of a token in its storage at r of its closest successors. The original bias elements are called masters. The bias elements storage is divided into two parts: the master storage Θ and the replica storage Ξ . Let $p.\Theta$ be the master storage of a peer p , while $p.\Xi$ be the replica storage of the peer p .

As their names suggest, the master storage stores the master copy of bias elements and the replica storage stores the replicas. The replicas holders know the location of the master copies of the replicas in their storage based on the r -neighbourhood information and the common knowledge that a bias element with key h_j is located at its immediate predecessor. This means a peer p stores the bias elements with key in

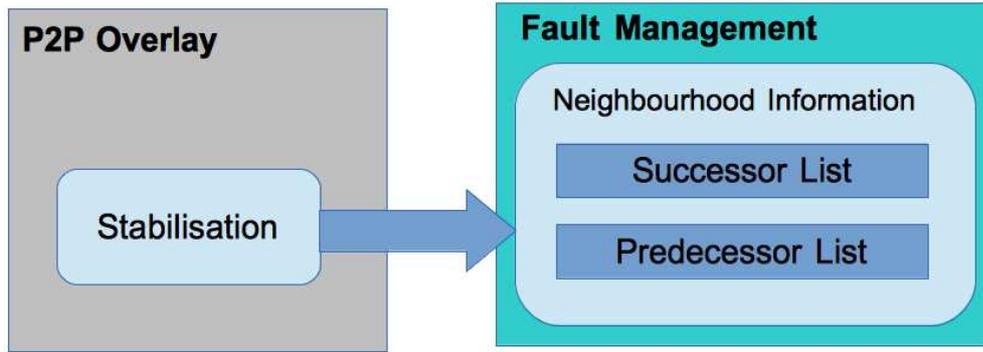


Figure 4.15: The neighbourhood watch service uses the information from the stabilisation function of the P2P overlay to obtain the neighbourhood information which includes the successor list and predecessor list.

the range of $[p.ID, p_{suc[0].ID})$ where $p.ID$ is the identifier of a peer p . While the master bias elements holders can trace their replicas by also using the r -neighbourhood information, the replicas are located at peers in r successor lists.

Let the i th successor of peer p is $p_{suc[i-1]}$ where the immediate successor of p is $p_{suc[0]}$. The i th successor of the peer p , $p_{suc[i-1]}$ holds the i th replica copy of a bias element k where peer p is the owner of the master copy of the bias element. For instance, $p_{suc[0]}$ stores the $\langle 1, k \rangle$ replica of the bias element k . In an example in Figure 4.16, a peer h_{50} holds the replica $\langle 1, h_{41} \rangle$ while peer h_{83} holds the replica $\langle 2, h_{41} \rangle$. This is summarised in Table 4.13.

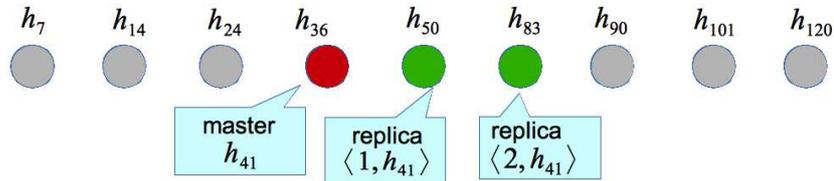


Figure 4.16: The master copy of h_{27} is placed at peer h_{36} and given the replication degree, $r = 2$, the replicas are located at peer h_{50} and peer h_{83} respectively

Only the master copy has a read-write permission; a replica is read-only to avoid the replica inconsistency which creates several replica versions which requires

Table 4.13: The placement of replicas for bias element h_{41} from the example in Figure 4.16.

Bias Element h_{41}		
Copy	Location	Peer Identifier
Master	p	h_{41}
Replica 1	$p_{suc[0]}$	h_{50}
Replica 2	$p_{suc[1]}$	h_{83}

a periodic update to achieve consistency. In case the master copy is unavailable at learning time (due to the failure of the peer which holds it), the query will be attended to by the successor peer which has a replica copy. Prior to storing the new training instance, the successor peer invokes the recovery process to replace the failed peer. The new owner of the master copy then learns the training instance via `localLearn(.)` function and updates the replicas at its first r successors via the `updateReplica(.)`. The `localLearn(.)` function is a function the P2P-GN (as described in Chapter 3), while `updateReplica(.)` is described as follows.

UpdateReplica(.) function Upon receiving the `REPLICA_UPDATE` message—which includes a replica copy, a peer x updates the received replica copy in its replica storage.

Figure 4.17 illustrates an example of the learning process for a bias element h_{41} based on the example in Figure 4.16. The peer h_{36} which has the master copy for h_{41} receives a learning request and performs the local learning (using `localLearn(.)` function). After completing the local learning, the peer h_{36} sends a replica copy of the updated version of bias element h_{41} to each of its closest r successors. The successors then update the copy of replica h_{41} in their replica storage.

Peer Selection Service

Peer selection service is useful to the convergecast approach, where m peers from the neighbourhood are selected by a peer p as its child nodes. The tree-structured

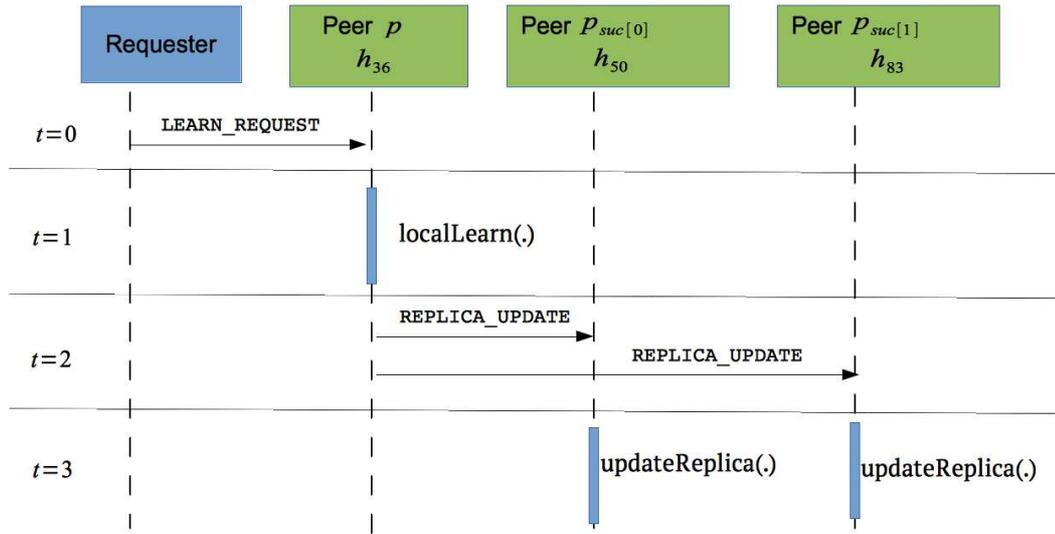


Figure 4.17: The sequence diagram for learning process for bias element h_{41} as in the example in Figure 4.16. Peer h_{50} and peer h_{83} are the replica holders, while, the peer h_{36} is the master copy holder.

convergecast approach faces the risk of losing information when internal nodes fail. These internal nodes may fail during or after aggregation in which case the parent node does not receive the aggregated result, or the internal nodes may fail after submitting the `FORWARD_REQUEST` or `LEARN_REQUEST` which results in the child nodes sending their predictions to the peer that no longer exists within the network. Therefore, the selection of the internal nodes is important to ensure the selected peers are fit enough to stay within the network during the query processing duration—from the time it receives the `FORWARD_REQUEST` query to the time it responds with the aggregated prediction in the `FORWARD_RESPONSE` message to the parent node.

Several algorithms have been proposed in the literature for peer selection which usually performed to select supernodes for multi layer P2P system [122, 98, 206]. Similar function applies here where the selected peers are expected to have better capability than others and stay long enough within the network. Nonetheless, our expectation for these selected peers may be not as high as the supernode selection since the period of query processing is relatively short and the peers are selected particularly for a single query. Moreover, internal node failure does not severely impact the computation in our work, since the processing can still proceed at the

requester end; instead it only delays the computation. Considering the high computational overheads of the available supernode selection algorithms, in this thesis, we use a simple peer selection procedure as follows.

Objective	: $\hat{\mathcal{P}} \leftarrow$ Select m peers from the set \mathcal{P}
Selection criteria	: $\tilde{\mathcal{U}} = \{\tilde{u}_1, \tilde{u}_2, \dots, \tilde{u}_k\}$
Minimum capability	: $\widetilde{\mathcal{U}}_{min} = \{\tilde{u}_{min,1}, \tilde{u}_{min,2}, \dots, \tilde{u}_{min,k}\}$
Criteria weight	: $\widetilde{\mathcal{W}} = \{\tilde{w}_1, \tilde{w}_2, \dots, \tilde{w}_k\}$

Our objective here is to obtain set $\hat{\mathcal{P}}$ by selecting m peers from set \mathcal{P} where \mathcal{P} represents all peers within a P2P network. The selection criteria is defined as set $\tilde{\mathcal{U}} = \{\tilde{u}_1, \tilde{u}_2, \dots, \tilde{u}_k\}$ where \tilde{u}_i is the i th selection criterion. The common set of criteria for peer selection is upload bandwidth, uptime, space and memory size, and CPU power. We also include the maximum connection for a given time per peer criterion, which is described in Appendix C. Set $\widetilde{\mathcal{U}}_{min}$ gives the minimum requirement for every criterion in $\tilde{\mathcal{U}}$, where $\tilde{u}_{min,i}$ is the minimum capability for \tilde{u}_i . The $\widetilde{\mathcal{U}}_{min}$ must be selected in a way that it is not too high, which may result in nothing from \mathcal{P} satisfying these minimum requirements. The criteria weight set $\widetilde{\mathcal{W}}$ defines the weight for every criterion. This weight indicates the importance of the criterion in selecting peers, where a higher weight represents a higher importance of the criterion, while, a lower weight represents a lesser important of the criterion compared to other criteria.

The peer selection procedure is described in the flow diagram in Figure 4.18. The set \mathcal{P} peers at first undergo the screening against the $\widetilde{\mathcal{U}}_{min}$. The peers which pass the screening, $\tilde{\mathcal{P}}$, are then included in a multi-criteria decision making process using the simple additive weighted (SAW) method [214] to give a score to the peers. The m peers that have the m th highest score are then selected into set $\hat{\mathcal{P}}$.

The SAW is used here considering its simplicity and it has been successfully applied in many problem domains [214]. The basic concept of SAW is to obtain the weighted sum of the performance for every option over all k attributes. The score

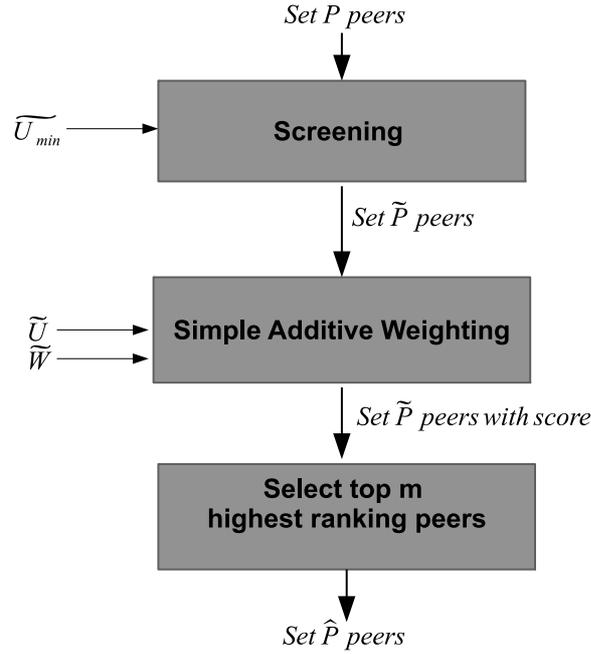


Figure 4.18: The peer selection process.

for peer p , $Score_p$ is calculated as follows.

$$Score_p = \sum_{i=1}^{k+1} \tilde{w}_i \tilde{v}_{p,i} \quad (4.15)$$

where \tilde{w}_i is the weight for criterion i , $0 < \tilde{v}_{p,i} \leq 1$, and $\tilde{v}_{p,i}$ is the normalised value of criterion i of a peer p . The value of $\tilde{v}_{p,i}$ needs to be normalised to ensure the comparable scale for the values.

Replica Recovery Service

Replica recovery is conducted to ensure high bias element availability and to maintain r replicas for every bias element. This includes controlling the number of replicas so that it always equals to r to avoid the flooding of unnecessary replicas and a total loss of replicas for a particular bias element. The replica recovery service is invoked in two ways:

- online recovery - when a peer joins or departs the network gracefully.

- periodic recovery - a change of neighbourhood is detected during a periodic recovery protocol after network stabilisation.

The online recovery of replicas is performed as soon as a peer requests to join or leave the network. In the case of a graceful departure, the departing peer informs its predecessor and transfers its master storage to the predecessor before leaving the network. Unlike abrupt peer failure, which is often detected after a periodic stabilisation process, the change of topology due to joining of peers and graceful departure of peers can be managed immediately.

In order to ensure the bias element availability in the system's lifetime, the placement of replicas is periodically updated. This includes replacing the master copy (when the peer that held the master copy departed), and maintaining r number of replicas in the system. The periodic recovery involves every peer within the network; it is conducted locally after the periodic network stabilisation. The network stabilisation—a protocol in the Chord overlay network—provides an information to the peers regarding their neighbourhood changes.

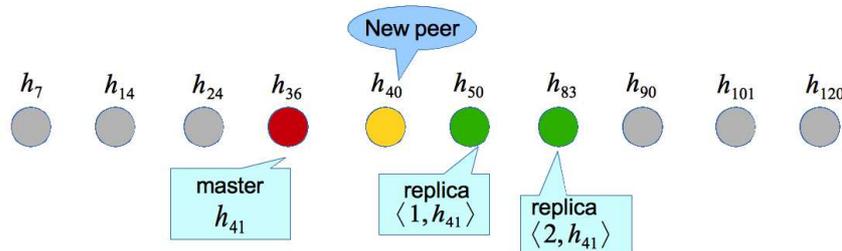
A recovery is executed at time t_{update} where $t_{update} = t_{lastupdate} + \delta$, where $t_{lastupdate}$ is the last periodic recovery time, and δ is a predefined time interval to update the replicas. The replica migration cost is relaxed by only performing the recovery process under certain conditions that involve neighbourhood changes. To monitor the changes, every peer also records the successor list and the predecessor list from the last update—*ex-successor list* and *ex-predecessor list*, respectively. Therefore, replicas are only sent to the peers that are new to the successor list—since other peers in the successor list already have the replicas.

Additionally, every peer also records all its new immediate predecessors since the last update in *inter-update predecessor list*. This information is crucial to correct any wrong placement due a peer joining and leaving in between updates, since this event is not detected by observing the changes to the predecessor and successor list since the last update. For example, at a peer p , its ex-predecessor list (from the recovery at $t = 0$) contains $\{2, 3, 4\}$ and its current predecessor list (during the recovery at

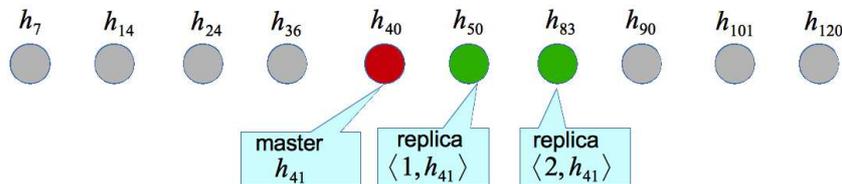
time $t = 1$) is $\{2, 3, 4\}$. Between the periodic recoveries in δ intervals, a new node 5 joins the network and it leaves the network after the online join recovery process has completed. Since there is no changes to the predecessor list (ex-predecessor list equals to the predecessor list), p does not perform any recovery process to correct the wrong placement caused by node 5 joining.

4.6.2 Recovering From The Aftermath of Peer Joining

In this section, we explain an online recovery process which is executed to recover the P2P-GN after the incorrect placement of bias elements due to peer joining. The joining of a peer p results in the incorrect placement of some master bias elements at $p_{pre[0]}$, since peer p may become the new successor of these bias elements (see Figure 4.19). In order to correct this, $p_{pre[0]}$ migrates the bias elements—that suppose to be located at p after a peer joining—to p . A peer joins the system by



(a) A peer h_{40} joins the network; results in an incorrect placement in the master bias element h_{41} location—the correct placement of h_{41} is at its new predecessor that is the peer h_{40} .



(b) Peer h_{36} which is the immediate predecessor of h_{40} transfers the master bias element h_{41} to the h_{40} through the $\text{joinRecovery}(\cdot)$ protocol.

Figure 4.19: An example of the joining node causing an incorrect placement in the master copy location and the $\text{joinRecovery}(\cdot)$ protocol being executed to correct this placement.

contacting one of the peers in the network—commonly known as a *bootstrapping*

peer. Let p be a joining node and p' be a bootstrapping peer, the joining procedure $\text{peerJoin}(p, p')$ is described in the following.

peerJoin(p, p') Initially, p' assists p by looking for the closest peer to p that is known to p' using $\text{findSuccessor}(\cdot)$ function. After establishing a connection with its immediate successor, p updates its finger table using updateFingerTable and then stabilises its neighbourhood using stabilisation . The $\text{findSuccessor}(\cdot)$, updateFingerTable and stabilisation are procedures in Chord network as can be found in Stoica et al. [179]. After stabilisation, p sends a **JOIN_P2P-GN** message to its neighbourhood ($\Gamma(p)$).

The **JOIN_P2P-GN** message triggers the execution of a $\text{onlineJoinRecovery}(\cdot)$ protocol at the recipient peers. This protocol is summarised in Protocol 4.4. Two main processes in this protocol are the update of neighbourhood information within $\Gamma(p)$ and the correction of the master bias elements' position.

Protocol 4.4. $\text{onlineJoinRecovery}$

$p \leftarrow$ the new joining peer.

At every peer $x := \{x \in \Gamma(p)\}$

upon receiving **JOIN_P2P-GN**

$x.\text{updateFingerTable}$.

$x.\text{stabilisation}$.

if x is equal to $p_{pre[0]}$ **then**

$\hat{T} \leftarrow \{g \in p_{pre[0]}. \Theta : g.\text{key} := [p.ID, p_{suc[0]}.ID]\}$.

if $\hat{T} \neq \emptyset$ **then**

$SList_{pre} \leftarrow$ the successor list of $p_{pre[0]}$

send **TRANSFER_ELEMENT** ($\hat{T}, SList_{pre}$) message to p .

end if

end if

At peer $x := \{x \in G_P\}$: **upon receiving** **TRANSFER_ELEMENT** ($\hat{T}, SList_{pre}$) **message**

store \hat{T} into $p.\Theta$

$SList \leftarrow$ the successor list of x

$NewSuc \leftarrow SList \setminus SList_{pre}$

for all $s \in NewSuc$ **do**

send **STORE_REPLICA** (\hat{T}) message to s .

end for

Upon receiving a **JOIN_P2P-GN** request from a new peer p , the receiving peers update their finger table and perform stabilisation. If the receiving peer is a peer $p_{pre[0]}$, it then responds by sending $\langle \hat{T}, SList_{pre} \rangle$ (via **TRANSFER_ELEMENT** message) to p where \hat{T} is a set of bias elements in its master storage ($p_{pre[0]}. \Theta$) with key within

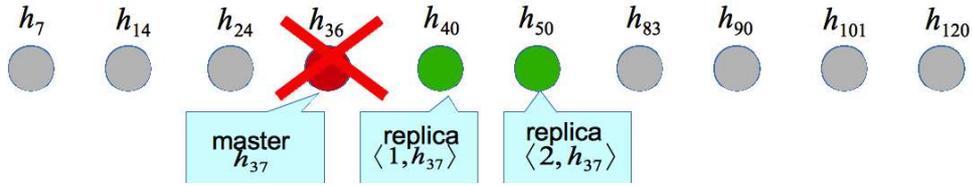
$[p.ID, p_{suc[0].ID}]$; $p.ID$ is an identifier for p and $SList_{pre}$ is the successor list of the peer $p_{pre[0]}$. Then, p receives \hat{T} and stores them into its master storage ($p.\Theta$). It also replicates \hat{T} at its r closest successors that are not in $SList_{pre}$ — these successors are denoted by $NewSuc$. $NewSuc$ is formally defined by $NewSuc := SList \setminus SList_{pre}$ where $SList$ is the successor list of p . By identifying $NewSuc$, we reduce the communication overhead in replication process from $\mathcal{O}(r)$ to $\mathcal{O}(|NewSuc|)$ where $|NewSuc| < r$.

In an example in Figure 4.19, a new peer h_{40} joins the network and subsequently, its predecessor, h_{36} transfers a bias element h_{41} from its master storage to the peer h_{40} via the `TRANSFER_ELEMENT` message. The peer h_{40} then stores the bias element in its master storage. h_{36} also provides its list of r successors to h_{40} ; h_{40} uses the list to identify any of its closest r successors that do not have replicas for the received bias elements from the h_{36} . In this example, no replication is required since the successor list of h_{36} equivalent to the successor list of h_{40} .

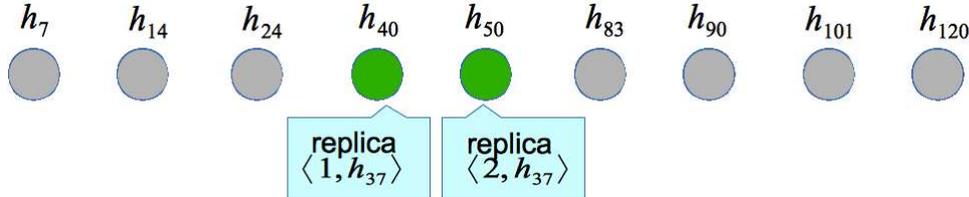
4.6.3 Recovering From The Aftermath of Peer Departure

This section discusses the proposed steps to undertake to ensure the P2P-GN recovers from the consequences of peer departures. A peer p leaves the network when it is disconnected due to several reasons, such as network failure, host failure or when the user quits the network. Figure 4.20 and Figure 4.21 illustrate two scenarios of how the system loses master copies and replicas, due to a peer departure within a neighbourhood of ten peers: $h_7, h_{14}, h_{24}, h_{36}, h_{40}, h_{50}, h_{83}, h_{90}, h_{101}$ and h_{120} . This is described as follows.

Scenario 1: The Loss of Master Copies Due to a Peer Departure In the first scenario, a peer h_{36} holds a master copy of h_{37} and since the replication rate is $r = 2$, two immediate successors of the h_{36} — a peer h_{40} and a peer h_{50} —holds the replicas of h_{37} . When the peer h_{36} left the network, the master copy of h_{37} is also lost and this results in the system has no master copy of h_{37} . Therefore, prior to



(a) The peer h_{36} left the network; together with the master bias element of h_{37} .



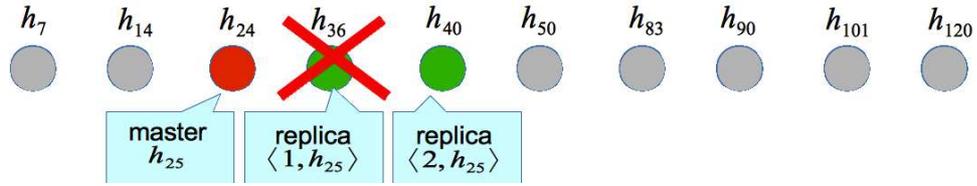
(b) There is no master copy of the bias element h_{37} within the network, however, there are two replicas of h_{37} at the successors of the peer h_{36}

Figure 4.20: An example of the scenario 1 where the system loses a master copy due to a peer departure. The replication degree in this example is two.

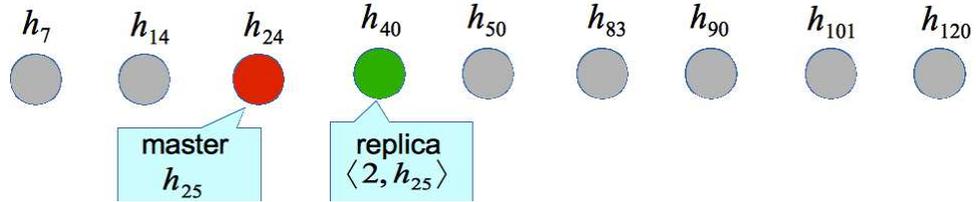
recovery, the system has to operate by using two available replicas at the peer h_{40} and the peer h_{50} .

Scenario 2: The Loss of Replicas Due to a Peer Departure In the second scenario, a peer h_{24} holds a master copy of bias element h_{25} , while a peer h_{36} and a peer h_{40} hold the replicas of h_{25} . As shown in Figure 4.21, one of the h_{25} 's replicas disappears together with the peer h_{36} . Hence, the system now only has one replica of h_{25} , but the required number of replicas per bias element is two (since $r = 2$) in this example.

We outline three events, which trigger the recovery process after a peer p leaves the network in Table 4.14. In the leave_event 1, a peer p failure is detected by its neighbours in $\Gamma(p)$ after **stabilisation** (a routine in the Chord overlay network system). A peer is aware of any changes to its neighbourhood; it knows if any peer within its neighbourhood has failed. Since these failures are usually noticed after the stabilisation phase, the recovery time may be delayed until the next stabilisation session in the leave_event 1. For instance, if a peer fail at time $t = 200$ seconds and **stabilisation** is executed for every 60 seconds, then the next recovery process will



(a) The peer h_{36} left the network; together with a replica bias element of h_{25} .



(b) The system only have one replica of the master bias element h_{25} due to the peer h_{36} departure.

Figure 4.21: An example of the scenario 2 where a peer departure results in the insufficient number of replicas available within the system. The replication degree, r in this example is two and so the system must preserve at least two replicas within the system.

Table 4.14: Events which trigger the recovering process following the peer p departure.

Recovery Process Triggering Events Due to the Departure of Peer p	
Event	Description
leave_event 1	the changes of the successor list and predecessor list at peers within $\Gamma(p)$. which are detected during periodically stabilisation phase.
leave_event 2	the learning request to a bias element which has no master copy due to the departure of p which held the master copy.
leave_event 3	the request to leave the network by a peer p in a smooth departure.

only start at $t = 240$ seconds, which results in the recovery delay being 40 seconds. Meanwhile, the system has to depend on the available replicas for its operation.

Another event which triggers the recovering process is the leave_event 2 when a peer receives a learning request from a requester for a bias element for which it holds the replica copy. Therefore, it notices that its immediate neighbour has failed since the master copy of the bias element must be located at its immediate neighbour. This triggers an immediate recovery of the master bias element. Among the three events, the leave_event 3 is the least risky since a leaving peer p quits the network gracefully by informing its neighbours about its departure so that a systematic recovery process can be executed—hence, avoiding loss of bias elements.

When a peer p fails, the system loses a number of bias elements and replicas along with this peer. However, replicas for bias elements from the master storage of p can still be found at its immediate r successors. Hence, these replicas can be used to recover the master storage of p . The abrupt failure can only be detected during the `leave_event 1` and `leave_event 2`. Hence, in addition to the online recovery process (which is executed immediately after a failure is detected) to handle the `leave_event 2` and `leave_event 3`, we also perform a periodic recovery process to handle the `leave_event 1`. The online recovery process is executed by the `onlineDepartureRecovery` protocol, while, the periodic recovery process is executed by the `routineDepartureRecovery`.

Online Departure Recovery

The `onlineDepartureRecovery` protocol is summarised in Protocol 4.5. During the

Protocol 4.5. `onlineDepartureRecovery`

```

1:  $p \leftarrow$  the leaving peer.
2:
3: At peer  $p_{pre[0]}$ : On leave_event 2
4: send FAILURE_DETECTED messages to  $\forall x \in \Gamma(p)$ .
5:
6: At peer  $p$ : On leave_event 3
7: send LEAVE_P2P-GN messages to  $\forall x \in \Gamma(p)$ .
8:  $\tilde{R} \leftarrow \{g \in p.\Theta : g.key = [p_{pre[0].ID}, p_{suc[0].ID}]\}$ .
9: if  $\tilde{R} \neq \emptyset$  then
10:   send TRANSFER_ELEMENT ( $\tilde{R}$ ) message to  $p_{pre[0]}$ .
11: end if
12:
13: At every peer  $x$  in  $\Gamma(p)$ 
14: upon receiving LEAVE_P2P-GN message and FAILURE_DETECTED( $p$ ) message
15:  $x.updateFingerTable$ .
16:  $x.stabilisation$ .
17:
18: At peer  $p_{suc[0]}$ : after receiving FAILURE_DETECTED message
19:  $\tilde{R} \leftarrow \{g \in p_{suc[0].\Xi} : g.key = [p_{pre[0].ID}, p_{suc[0].ID}]\}$ .
20: if  $\tilde{R} \neq \emptyset$  then
21:   send TRANSFER_ELEMENT ( $\tilde{R}$ ) message to  $p_{pre[0]}$ .
22: end if
23:
24: At peer  $x := \{x \in G_P\}$ : upon receiving TRANSFER_ELEMENT ( $\tilde{R}$ ) message
25: store  $\tilde{R}$  into  $x.\Theta$ 

```

`leave_event 2`, the predecessor of the failed peer p (i.e., $p_{pre[0]}$) informs the neighbours of p (i.e., $\Gamma(p)$) regarding the failure event via a `FAILURE_DETECTED` message. Every peer in $\Gamma(p)$ then performs stabilisation and corrects its finger table and neighbours'

information. Peer $p_{suc[0]}$ then sends a set of replicas from its replica storage ($p_{suc[0]}.\Xi$) with keys $[p_{pre[0]}.ID, p_{suc[0]}.ID)$ to $p_{pre[0]}$ through a `TRANSFER_ELEMENT` message.

During a `leave_event 3`, a peer p leaves the network smoothly in a way that it alerts its $\Gamma(p)$ neighbourhood (via `LEAVE_P2P-GN` message) about its departure. This allows an easy and smooth transition; the peer p sends a set of master copies from its storage, \tilde{R} , through a `TRANSFER_ELEMENT` message to peer $p_{pre[0]}$ (its immediate predecessor) for storing before quitting the network. \tilde{R} is a set of master bias elements in $p.\Theta$ that have keys in $[p_{pre[0]}.ID, p_{suc[0]}.ID)$. Upon receiving \tilde{R} , $p_{pre[0]}$ stores them in its master storage ($p_{pre[0]}.\Theta$). Meanwhile, `updateFingerTable` and `stabilisation` are performed by every peer in $\Gamma(p)$ to correct their neighbourhood information upon receiving the `FAILURE_DETECTED` and the `LEAVE_P2P-GN` messages.

Routine Recovery Approach

It is crucial to maintain the performance of the P2P-GN throughout the system lifetime. In the P2P-GN, this is done by periodically correcting the wrong placement and replacing the lost master or replica copy. We devise the protocol `routineRecovery` as described in Protocol 4.6 for this task and it is performed following a periodic Chord stabilisation.

The `routineRecovery` is executed throughout the system lifetime within a predefined time interval δ . Every peer x in the network G_P executes this protocol locally. However, only peers who detect changes to their neighbourhood communicate with others to collaboratively recover the lost bias elements. Therefore, when the network is in a stable state (no churn), no communication is required at all. The inputs to Protocol 4.6 are given in Table 4.15 and these are recorded by every peer. The *SList* and *PList* are obtained after stabilisation, while, *SList'* and *PList'* are updated in the last execution of this protocol. *AList* is recorded throughout the period in between the last execution of this protocol and the current execution.

The execution of a recovery process is subject to two conditions: first, the change of the predecessor list \mathcal{C}_{pre} (defined by Equation 4.16); and second, the change of

the successor list \mathcal{C}_{suc} (defined by Equation 4.22). The recovery process is only performed when any of these conditions is true. By doing so, we significantly reduce the communication costs for the system recovery.

Protocol 4.6. routineRecovery

```

1:  $x \leftarrow$  current node where  $\{x \in G_P\}$ 
2: Input:  $SList', SList, PList', PList, AList$ 
3:
4: recoveryOnCondition $\mathcal{C}_{pre}$  ( $PList', PList, AList$ )
5: recoveryOnCondition $\mathcal{C}_{suc}$  ( $SList', SList$ )
6:
7: upon receiving TRANSFER_ELEMENT ( $\hat{C}$ ) message
8: store  $\hat{C}$  master bias elements.
9:
10: Upon receiving STORE_REPLICA ( $\hat{C}$ ) message
11: store the received  $\hat{C}$  replicas.
12:
13:  $PList' \leftarrow PList$ 
14:  $SList' \leftarrow SList'$ 
15: cleanReplicaStorage(.)
  
```

Table 4.15: Input to the routineRecovery Protocol

Notation	Description	Notation	Description
$AList$	inter-update predecessor list	x	current node
$SList'$	ex-successor list.	$SList$	successor list.
$PList'$	ex-predecessor list.	$PList$	predecessor list.

For the first condition \mathcal{C}_{pre} (as described in Algorithm 4.7), a peer p identifies the failed peers to recover the lost bias elements. After these failed peers have been identified, then the range of keys which were assigned to these peers then can be discovered for recovery. First, the peer p checks the changes to its predecessor list by comparing the current predecessor list $PList$ with the ex-predecessor list $PList'$. Any peer in $PList'$ that is not in $PList$ is included in a set of obsolete predecessors $ObsPre$. Second, p compares the inter-update predecessor list $AList$ with $PList$. These steps are executed to identify predecessors that have failed before the current update. We denote these peers by $DeadA$ —peers that are in $AList$ but not in $PList$. If the combined set of $DeadA$ and $ObsPre$ is not empty, this means that there are failed peers to recover, and the condition \mathcal{C}_{pre} is true—this causes the recovery process to proceed.

Protocol 4.7. $\text{recoveryOnCondition}_{C_{pre}}(PList', PList, AList)$

```

1:  $x \leftarrow$  current node where  $\{x \in G_P\}$ 
2:
3:  $DeadA \leftarrow AList \not\subseteq PList$ 
4:  $ObsPre \leftarrow PList' \not\subseteq PList$ 
5:  $MPre \leftarrow PList \cap PList'$ 
6:  $RPre \leftarrow \{s \in ObsPre; z \in MPre; s, z \in PList' : s.index > z.index\}$ .
7:  $JPre \leftarrow ObsPre \not\subseteq RPre$ 
8: if  $C_{pre}$  is true then
9:   for all  $\widehat{pre} \in PList$  do
10:      $\widehat{C} \leftarrow \emptyset$ 
11:     for all  $s \in DeadA$  do
12:        $\widehat{C} \leftarrow \widehat{C} \cup \text{getLostContent}(s, x, x.\Xi)$ 
13:     end for
14:     for all  $s \in JPre$  do
15:        $s_{suc[0]} \leftarrow \{a \in MPre' : a.index := s.index - 1\}$ 
16:        $\widehat{C} \leftarrow \widehat{C} \cup \text{getLostContent}(s, s_{suc[0]}, x.\Xi)$ 
17:     end for
18:     if  $\widehat{C} \neq \emptyset$  then
19:       send TRANSFER_ELEMENT ( $\widehat{C}$ ) message to  $\widehat{pre}$ .
20:       for all  $s \in SList_{\widehat{pre}}$  do
21:          $\widehat{pre}$  sends STORE_REPLICA ( $\widehat{C}$ ) message to  $s$ .
22:       end for
23:     end if
24:   end for
25: end if

```

Condition C_{pre} Let $ObsPre$ (defined by Equation 4.17) be a set of expired predecessors from ex-predecessor list ($PList'$) which are not in the current predecessor list ($PList$) and $DeadA$ (defined by Equation 4.18) be a set of inter-update predecessor list ($AList$) which is not in the current predecessor list ($PList$). The first recovery condition, C_{pre} is true when $ObsPre$ and $DeadA$ is not empty set. This is formally defined in Equations 4.17, 4.18 and 4.16 as follows.

$$C_{pre} = \begin{cases} \text{true,} & \text{if } ObsPre \cup DeadA \neq \emptyset \\ \text{false,} & \text{otherwise} \end{cases} \quad (4.16)$$

where

$$ObsPre := PList' \not\subseteq PList \quad (4.17)$$

$$DeadA := AList \not\subseteq PList \quad (4.18)$$

If condition C_{pre} is true then we start the recovery process as describe in Line 10 to Line 26 of Algorithm 4.6. The recovery requires two sets of peers: $ObsPre$ and alive peers in $PList'$ (denoted by $MPre$). Successors of peers in $ObsPre$ can be

obtained from $PList'$, while, $MPre$ can be obtained from the intersection of $PList'$ and $PList$ (see Equation 4.19). Next, we identify a set of peers $RPre$ which are failed peers in $ObsPre$ that have at least one alive successor (see Equation 4.20). Following this, we identify $JPre$ which are peers in $ObsPre$ but not in $RPre$ (see Algorithm 4.21). The idea is that if an alive successor x performs recovery for a failed peer p , then the current peer does not need to recover p . Hence, we can further eliminate the redundant recovery process here, as well as reducing the communication overheads. This is summarised in the following Equations 4.19, 4.20 and 4.21.

$$MPre := PList \cap PList' \quad (4.19)$$

where $MPre$ is a set of predecessors which are still alive after the last recovery session.

$$RPre := \{s \in ObsPre : s.index > z.index, z \in MPre\} \quad (4.20)$$

where $RPre$ is a set of successors of failed predecessors which are still alive after the last recovery session, $s.index$ refers to the index of s in $PList'$ and $z.index$ refers to the index of z in $PList'$ ^{4.10}.

$$JPre := ObsPre \setminus RPre \quad (4.21)$$

Now, the lost master bias elements that need to be recovered are identified. Function `getLostContent(.)` is used to identify the lost bias elements with a failed peer. It is described as follows.

getLostContent($s.ID, s_{suc[0]}.ID, x.\Xi$) Get a set of replicas in $x.\Xi$ to be recovered at peer s where replica keys are in $[s.ID, s_{suc[0]}.ID)$.

The closest successor of a failed peer is required as an input to `getLostContent(.)`. The closest successor of $s \in JPre$ can be identified from $MPre$ by identifying a peer in $MPre$ which has index smaller than s . This index refers to the index in

^{4.10}Given an ex-predecessor list $PList' := \{p_{pre[0]}, p_{pre[1]}, \dots, p_{pre[r-1]}\}$, the index of $p_{pre[y]}$ is y

$PList'$ where $s, s_{suc[0]} \in PList'$. To discover the lost bias elements with the failed peers in $DeadA$, the current peer is assumed to be the closest successor for every $s \in DeadA$ since the successor is unknown to the current peer's knowledge.

Initially, the bias elements that are lost with failed peers in $DeadA$ are discovered (see Line 13 of Algorithm 4.6). Then, we also identify the bias elements that are lost with peers $JPre$ (see Line 17 of Algorithm 4.6). In case the resulting bias elements to be recovered at a predecessor $\widehat{pre} \in PList$ —referred as \widehat{C} —are larger than zero, then the \widehat{C} bias elements are sent to \widehat{pre} for storing via a **TRANSFER_ELEMENT** message. This provides another restriction on the communication of the current peer with its r closest predecessors where the communication to a peer $\widehat{pre} \in PList$ can be avoided if \widehat{C} is empty. After storing the \widehat{C} bias elements in its storage, the \widehat{pre} sends a **STORE_REPLICA** message that contains \widehat{C} to its r closest successors for replication.

Under the second condition (\mathcal{C}_{suc}) (as described in Algorithm 4.8), a peer x detects the change of its successor list, particularly the change of the first r successors in the list. Let $SList'$ denotes a set of first r successors from the successor list and let $SList$ denotes a set of first r successors from the current successor list. The peer p_{pre} obtains a set of successors in the $SList$ which are not elements in the $SList'$ and denotes this set by $ChangedSuc$. This set represents the current first r successors that are newly added to the first r successor list. Then, it sends all replicas of its master bias elements to all peers in $ChangedSuc$ via a **STORE_REPLICA** message and these peers store these replicas in their storage.

Protocol 4.8. $recoveryOnCondition_{\mathcal{C}_{suc}}(SList', SList)$

```

1:  $x \leftarrow$  current node where  $\{x \in G_P\}$ 
2:  $ChangedSuc \leftarrow SList \not\subseteq SList'$ 
3: if  $\mathcal{C}_{suc}$  is true then
4:    $x.\Theta \leftarrow$  all master bias elements in  $x$ .
5:   for all  $s \in ChangedSuc$  do
6:     send STORE_REPLICA ( $x.\Theta$ ) message to  $s$ .
7:   end for
8: end if

```

Condition \mathcal{C}_{suc} The second recovery condition \mathcal{C}_{suc} is defined as follows.

$$\mathcal{C}_{suc} = \begin{cases} \text{true,} & \text{if } ChangedSuc \neq \emptyset \\ \text{false,} & \text{otherwise} \end{cases} \quad (4.22)$$

where

$$ChangedSuc := SList \not\subseteq SList' \quad (4.23)$$

After the recovery is completed, the peer updates its ex-successor list and ex-predecessor list to the current successor list and current ex-predecessor list, respectively. The final task in the routine recovery is the replica storage clearing which will be discussed next. This aims to remove the replicas which should no longer be stored at the peer.

Replica Storage Cleaning Although online and periodic recovery maintain the availability of master and replica copies of bias elements in the system, these result in a large number of unnecessary replicas in the network. The number of replicas grows over time and this increases the storage overheads per peer. Therefore, a function `cleanReplicaStorage(.)` is included in the routine recovery protocol to clean the storage at every peer, so that the number of replicas for every bias element is maintained as an integer r , where r is the replication degree and $r > 0$. This is done locally and therefore, no communication is required.

cleanReplicaStorage(.) Get a set of replicas in x 's replica storage, $x.\Xi$ where replica keys are not in $[x_{pre[r-1]}.ID, x.ID)$ and delete these replicas from $x.\Xi$.

In the function `cleanReplicaStorage(.)`, every peer x identifies keys in its replica storage in range $[x_{pre[r-1]}.ID, x.ID)$. A peer $x_{pre[r-1]}$ is the last predecessor in the predecessor list of the peer x . Keys that are outside of this range are considered expired and so these keys are deleted from the replica storage. By doing so, the

unnecessary copies of the replicas are disposed from the system—to avoid wasting resources; thus, the number of replicas for every bias elements is maintained.

4.7 Evaluating System Reliability

In this section, we aim to investigate the effectiveness of the proposed fault-tolerant procedure in maintaining the high rate of bias element availability under churn. We also measure the resulting communication overheads which include the number of communications and the mean communication load per message. Here, we show that the network stability, despite the presence of a heavy churn, is efficiently maintained by the integration of the underlying structured P2P services with our proposed recovery scheme.

Initially, we explain the simulation set up in Section 4.7.1 and then, we explain the lifetime churn model that was used in this thesis in Section 4.7.2. In the first part of our experiment, we demonstrate the effect of dynamic networks on the P2P-GN performance prior the adoption of any fault-tolerance mechanism. We conducted an experiment to analyse the effect of joining and leaving nodes on the bias element availability and on the correctness of the P2P-GN within dynamic networks. This is followed by evaluating our proposed fault tolerance scheme in improving the reliability of the P2P-GN in the second part of the experiment.

4.7.1 Simulation Set-Ups

The Waveform data generator (Version 2) from the UCI database [13] were used to generate 5,000 training instances and 500 test instances. This dataset has 40 attributes and 3 classes. A small number of instances is sufficient enough, since our major aim here is to show the effectiveness of our fault-tolerant approach in handling the dynamic network.

We used the event based Peersim simulator [136] to simulate the Chord P2P network and implemented the P2P-GN algorithm on this platform. The simulation

was run on a PC with Intel Core i-7 2.9GHz and 8Gb memory. Particularly for this experiment, we coded a module that implements a dynamic network using a Weibull lifetime churn model [139] which will be discussed Section 4.7.2. The fault-tolerant approach, which has been proposed in this section, was also implemented in a separate module from the P2P-GN implementation within the simulator. As the dynamic network module, the periodic recovery module (which implements the `routineDepartureRecovery` protocol) was executed at a predefined time interval.

4.7.2 Churn Modelling

The dynamic network environment in this chapter was modelled using lifetime churn with Weibull distribution, as it was shown in the research in Aggarwal et al. [2] to reflect the real P2P users' behaviour. The lifetime churn model that is used to

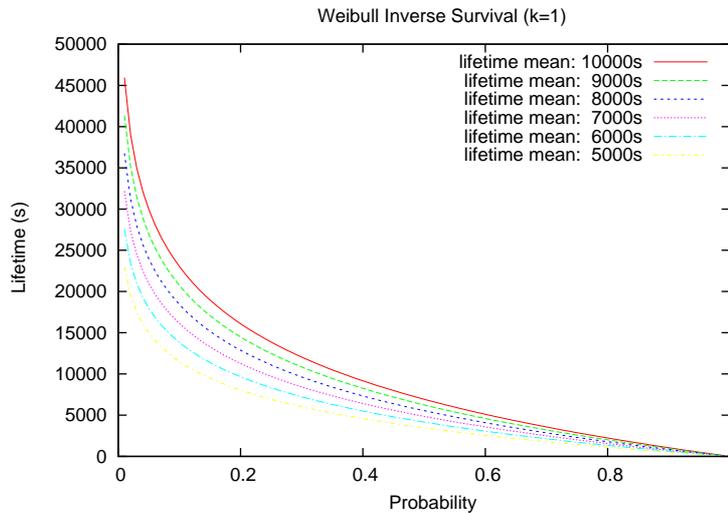


Figure 4.22: The lifetime VS probability plot for the Weibull inverse survival function with parameter $k = 1$ and $MTTF = \{5,000, 6,000, 7,000, 8,000, 9,000, 10,000\}$ seconds.

simulate the nodes' joining time and lifetime is described in Figure 4.22. Using a shape parameter $k = 1$, we varied the Mean-Time-to-Fail ($MTTF$) from 10,000 seconds to 5,000 seconds—this can be considered as an intense churn rate. The Weibull inverse survival function was used to estimate the probability failure time

as follows [139]:

$$\mathcal{Z}(p) = \tilde{\lambda}(-\ln(p))^{1.0/k} \quad (4.24)$$

where $\tilde{\lambda} = MTTF/\Gamma(1 + \frac{1}{k})$ and $k > 0$ is a shape parameter, while $1 \geq p > 0$ is a random number which represents the probability value. Note that churn rate increases with a decrease of the value a .

4.7.3 Investigating The Churn Effect on the P2P-GN Performance

The availability of a bias element ρ_A at time t depends on the probability of a peer p (which stores ρ_A) being alive at time t . To investigate the effect of leaving and joining peers to the P2P-GN performance, we ran a series of experiments in evaluating its performance under an extreme churn. In these experiments, we were particularly interested to find out the following:

- the changes in network topology under an extreme churn;
- the effect of an increase in churn rate (increase in $MTTF$ of the churn model) on the bias element availability;
- the effect of the network topology changes and the bias element availability on the P2P-GN accuracy; and
- the effect of churn on the P2P-GN performance when the network size is large.

Churn Effect on The Network Topology

We ran simulations to explore the changes in network topology caused by different $MTTF$ values. The next experiment aims to show the level of network disruption which is handled by our proposed recovery solution. The outcome of this simulation is shown in Table 4.16.

The peer join rate and the peer depart rate per second are presented here. The peer join rate is the total number of peers joining the network divided by the total

Table 4.16: Topology changes in a period 20,000 seconds using Weibull churn model with different $MTTF$.

Event	$MTTF$ (in seconds)					
	5,000	6,000	7,000	8,000	9,000	10,000
Peer Join Rate (per seconds)	0.2649	0.2096	0.1670	0.1381	0.1149	0.0957
Peer Depart Rate (per seconds)	0.2091	0.1573	0.1218	0.0964	0.0809	0.0685

simulation time. The peer depart rate is the total number of peers leaving the network divided by the total simulation time. The first 1,000 seconds of simulation time was the time taken for the network formation. Hence, the total simulation time for the joining rate and leaving rate here is 19,000 seconds.

We found that a total of 5,034 peers departed from the network and 3,973 peers joined the network within the period of simulation time $t = 0$ to $t = 20,000$ seconds under churn with $MTTF = 5,000$ seconds. This means the peer joining rate is 0.2649 per second and the peer leaving rate is 0.2091 per second. This also means that one peer joins the network every 3.77 seconds, and one peer leaves the network every 4.78 seconds, simulating a very high churn.

Churn Effect on The Bias Element's Availability

In the next experiment, we investigate the effect of churn on bias elements availability. The simulated network has initially 500 peers; the network size is small enough to evaluate the effect of churn. Given Proposition 4.5 of Section 4.5, it is expected that a large network size—which is usually the case of the P2P networks—provides greater robustness to the P2P-GN. Figure 4.23 describes the effect of the heavy churn with $MTTF = 5,000$ seconds on the bias elements availability which were recorded in snapshots between simulation time 0 to 7,000 seconds.

The bias element availability degrades since a leaving node may result in losing the queried bias element. The distributed memories which are stored at many different peers may eventually all be lost when these peers die over time. As a bias element is physically located within a peer, it moves together with the peer when the peer changes a position due to the joining and leaving peers. This leads to an

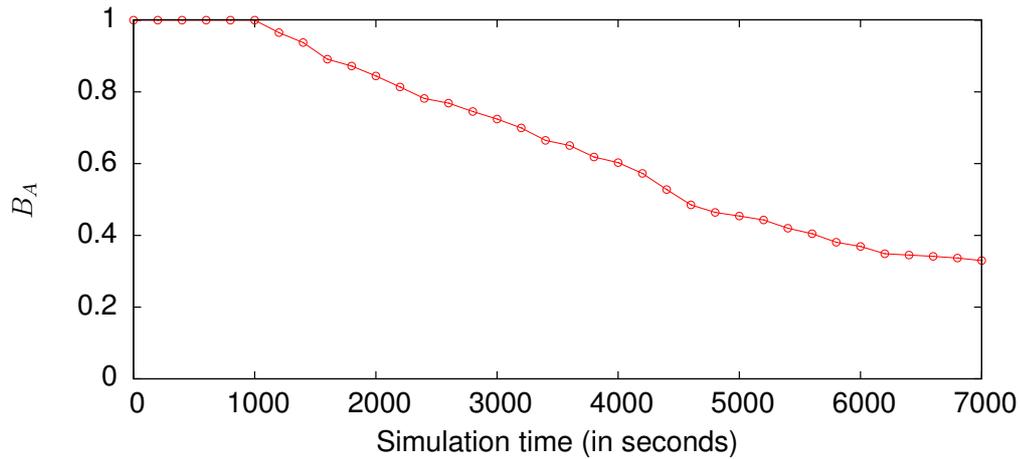


Figure 4.23: The decreasing bias element availability within the recorded time from 0 to 7,000 seconds under the lifetime churn with mean 5,000 seconds.

incorrect placement, as any bias element with identifier $[p.ID, p_{suc[0]}.ID)$ should be located at the peer p where $p.ID$ is the peer identifier. The incorrect placement results in the bias element becomes hard to locate and this usually incurs a high latency for retrieval. In the worst case, the data may be unable to be retrieved at all.

Churn Effect on The P2P-GN Correctness

Next, we conducted an experiment to evaluate the effect of churn to the correctness of the P2P-GN, as well as investigating the correlation of bias element availability with the accuracy. The lifetime mean in the churn model was set to 5,000 seconds and recorded the results at several different snapshots. Each snapshot was recorded at every time t_q . In this experiment, the learning was completed before the churn started at time $t = 0$ and the recall queries were submitted for every 20 seconds started from $t = t_q$. We froze the network at time $t = t_q$ (stopping all the joining and leaving processes after time $t = t_q$) to investigate the effect of the leaving and joining peers from $t = 0$ to $t = t_q$ on the bias elements availability and the P2P-GN accuracy. Note that, in case of joining peers, a number of bias elements are

migrated to the new joining peer, provided the peer is responsible for storing these bias elements.

The result is reported in Table 4.17. The bias element availability at time t_q

Table 4.17: The accuracy with bias element availability under Weibull churn model with $MTTF = 5,000$ seconds at different points of simulation time.

Snapshots t_q (seconds)	$B_A[t_q]$	Accuracy
1,000	1	0.7604
2,000	0.8487	0.7624
3,000	0.7251	0.7165
4,000	0.6029	0.7265
5,000	0.4537	0.6387
6,000	0.3693	0.6028
7,000	0.3296	0.5449

is denoted by $B_A[t_q]$. We found that, at $t_q = 1,000$ seconds, the bias element availability is 1 since no peers leave the network and the accuracy is 0.7604 which also represents the accuracy in the network with no churn (we refer this as an optimal accuracy henceforth in this chapter). When t_q increases, then the bias element availability and accuracy decreases slowly. At $t_q = 7,000$ seconds, the bias element availability is very low at only $\approx 33\%$ from the original number of bias elements (the number of bias elements after the learning is completed). However, the accuracy is reasonably high with only 0.2155 loss from the optimal accuracy. This reflects the ability of the P2P-GN to make correct predictions even under a stressful state; operating with insufficient information.

In the previous experiment, we used churn model with $MTTF = 5,000$ seconds. Hence, we aim to show the effect of different $MTTF$ s on the bias elements availability and the P2P-GN accuracy after stopping churn at simulation time $t_q = 7,000$ seconds, by varying the lifetime mean of our churn model from 5,000 seconds to 10,000 seconds. We present the outcome in Table 4.18; the low $MTTF$ in the Weibull churn model severely affects the P2P-GN operation with low bias element availability in comparison to the higher $MTTF$. Therefore, by using a churn model

with low $MTTF$ that equals 5,000 seconds, as in the following experiments in Section 4.7.4 illustrate, we have sufficiently proven the effectiveness and efficiency of our fault tolerance scheme

Table 4.18: The bias element availability and accuracy after $t_q = 7,000$ seconds under different $MTTF$.

$MTTF$ (seconds)	$B_A[t_q]$	Accuracy
5,000	0.3296	0.5449
6,000	0.3693	0.6107
7,000	0.4352	0.6447
8,000	0.4689	0.6567
9,000	0.5483	0.6766
10,000	0.6029	0.7025

The outcome of this experiment clearly proves the predicted linear correlation of the bias element availability and the system accuracy—when the bias element availability increases, the accuracy also increases. Therefore, this led us to use the bias element availability metric to evaluate the performance of the P2P-GN, along with the proposed fault-tolerance approach.

4.7.4 Evaluating the Performance of Fault Tolerance Scheme

Here, the performance of the proposed fault tolerance approach is evaluated against its ability to improve the bias element availability under heavy churn ($MTTF = 5,000$ seconds) and its efficiency to maintain the P2P-GN effectiveness. The results from the previous experiments (see Section 4.7.3) provide a benchmark in selecting the simulation parameters to evaluate our fault-tolerance scheme. We can infer that the performance of our scheme is better than the reported results in this section when the $MTTF$ is larger than 5,000 seconds. We can also infer that the accuracy of the P2P-GN is the same as that of implementation within a stable network when the bias element availability equals to 1. Therefore, the simulation in this experiment was run under a churn model with $MTTF = 5,000$ seconds and the starting network

size was set at 500 peers. Our target bias element availability to be achieved by the proposed fault tolerance scheme is 1.

Eight metrics were used to evaluate our fault tolerance scheme performance:

- the availability rate of master bias elements, MB_A ;
- the availability rate of replica bias elements, RB_A ;
- the number of messages per second, M/t ;
- the data transferred per second, L/t ; and
- the average number of messages per peer for every routine recovery, $M_{routine}/p$.

These metrics are calculated as follows:

$$MB_A = \frac{n_{MB}}{n_{OB}} \quad (4.25)$$

$$RB_A = \frac{n_{RB}}{n_{OB} \times r} \quad (4.26)$$

where n_{MB} is the available number of master bias elements in the system, n_{OB} is the original number of bias elements (the number of bias elements in a stable network), n_{RB} is the available number of replicas in the system and an integer r is the replication rate and $r > 0$.

$$M/t = \frac{n_M}{t_{total}} \quad (4.27)$$

$$L/t = \frac{t_{load}}{t_{total}} \quad (4.28)$$

where t_{total} is the total simulation time, n_M is the total number of messages within t_{total} , and t_{load} is the total size of data transferred (in MB) within t_{total} .

$$M_{routine}/p = \frac{M_{routine}}{N} \quad (4.29)$$

where $M_{routine}$ is the number of messages in an execution of a `routineRecovery(.)`. The metric $M_{routine}/p$ gives the number of messages that are consumed per peer

in every `routineRecovery(.)` execution. The number of messages and message load in the execution of `routineRecovery(.)`, protocol were recorded to demonstrate the overheads of `routineRecovery(.)` which is the main operation to maintain the system availability within dynamic networks.

When MB_A is high (approaching 1) then the overall bias element availability is also high. RB_A provides a measurement on the number of replicas in the network. When RB_A is higher than the replication rate r , then we know that there are extra replicas and therefore it is not storage-efficient. When the RB_A is lower than r , it shows that the number of backups is not sufficient and if this value keeps dropping then the system may end up losing some bias elements (the master copies and replica copies). The fault tolerance scheme is effective in maintaining the accuracy of the P2P-GN in a stable state (the state without churn) when the MB_A equals to one. If the RB_A equals to r , then the fault tolerance scheme is storage-efficient and effective to maintain the correct number replicas per bias element.

Results

We show that our fault-tolerance scheme is able to improve the bias element availability of the P2P-GN operating under a heavy churn— $MTTF = 5,000$ seconds. This, however, is subject to the length of interval between routine recovery which is denoted by δ and the replication rate r . We varied the δ value from 100 seconds by doubling the value until $\delta = 800$ seconds. This range of interval is low but high enough to evaluate our fault tolerance approach. We also varied the r values from 2 to 5 since our preliminary experiment showed that $r = 5$ is sufficient to maintain the system in a long period of simulation time (at least 20,000 seconds in this experiment). Then, we explore the fault-tolerance performance by decreasing the r value to 2. It can be assumed that if r is larger than 5, a better reliability can be achieved.

The result in Figure 4.24 shows that if $\delta = 100$ seconds, $r = 3$ is sufficient to maintain the bias element availability. However, when δ is higher, more replicas

are required. Our result shows that if $\delta = 800$ seconds (which is considered high), replication rate of 5 is sufficient to preserve the bias element availability.

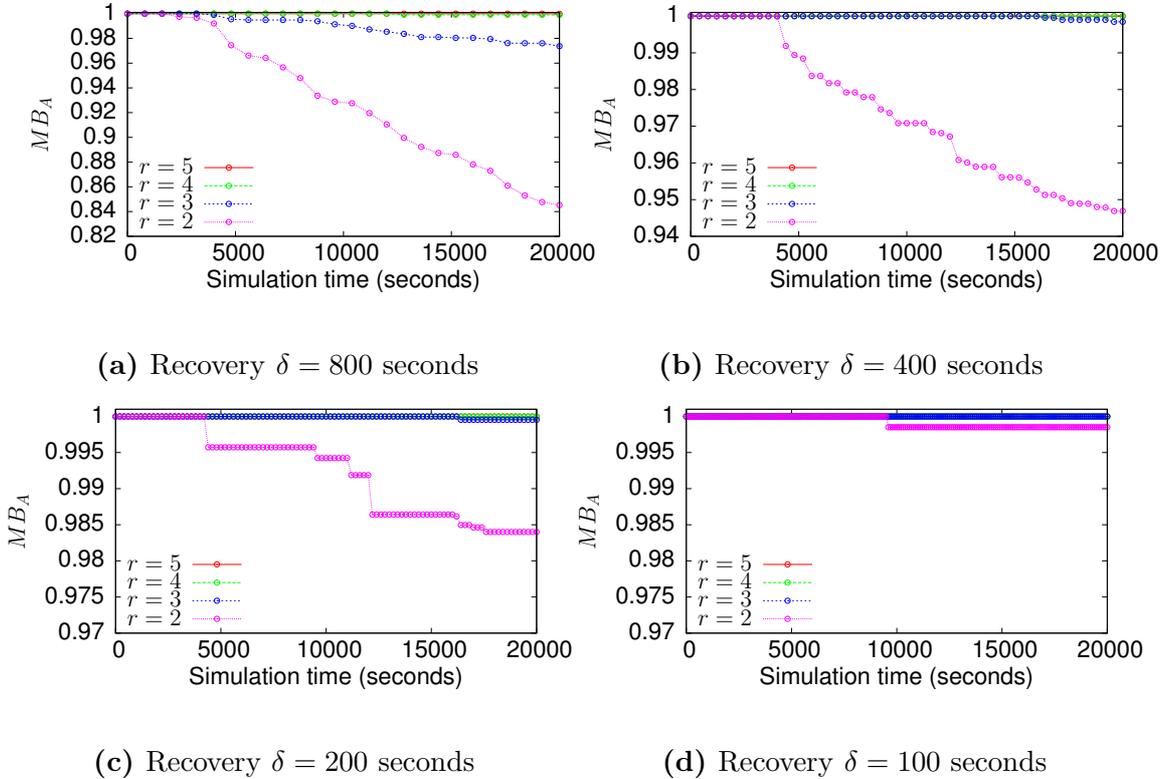


Figure 4.24: MB_A with recovery $r = \{2, 3, 4, 5\}$ and $\delta = \{100, 200, 400, 800\}$ under churn with lifetime mean 5,000 seconds within 20,000 seconds simulation time.

As shown in Figure 4.25, RB_A never grows more than r in any of the experiments. This shows that the function `cleanReplicaStorage(.)` effectively controls the number of replicas within the networks. This effectively suggests the storage-efficiency of our fault tolerance scheme. The number of replicas (RB_A) can be easily maintained high when r is high and δ is small. In our experiment, when $r = 5$ and $\delta = 100$, the RB_A was consistently high. In Figure 4.25, we can see that RB_A dropped faster for large δ than for small δ . During our experiment within 20,000 seconds simulation time, we found that, for high $\delta = 800$ seconds, the RB_A declined significantly when r is equal or smaller than 3 and it declined in slower pace when $r = 4$. All in all, if we decide to use a small number of replicas, then it is important to ensure that δ is small enough to maintain the performance of the P2P-GN.

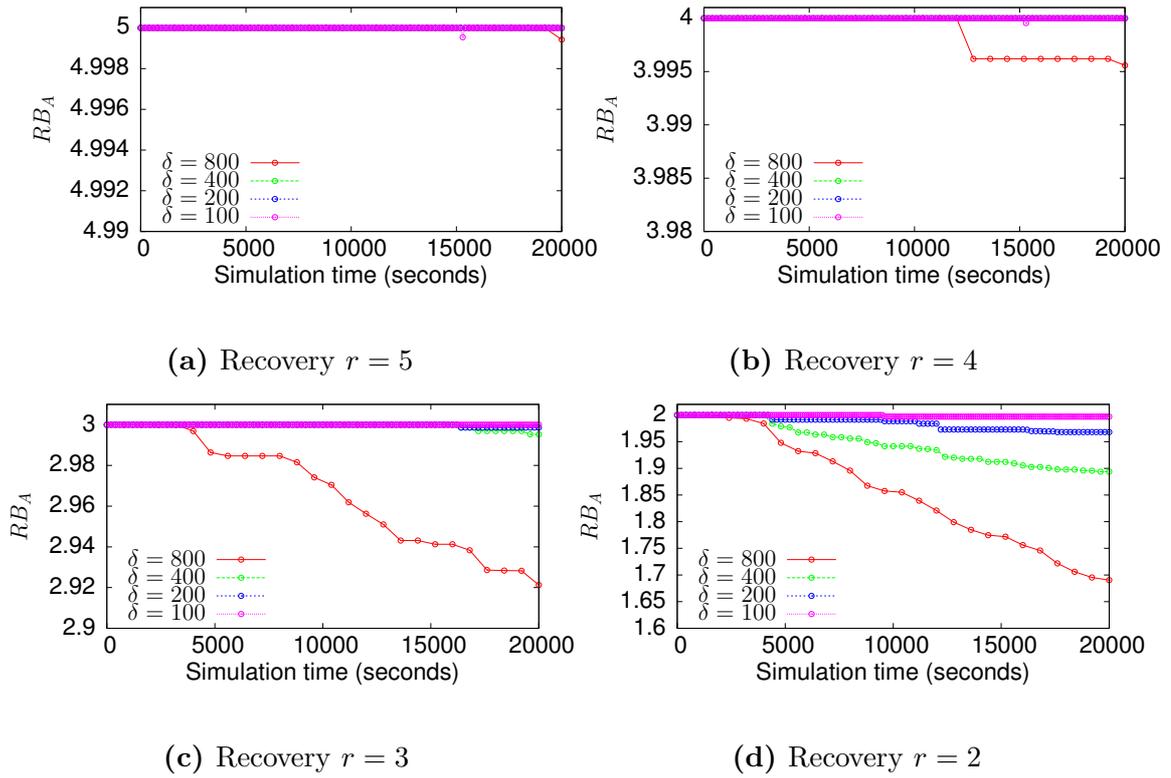


Figure 4.25: RB_A with recovery $r = \{2, 3, 4, 5\}$ and $\delta = \{100, 200, 400, 800\}$ under churn with lifetime mean 5,000 seconds within 20,000 seconds simulation time.

Although a high r and a short δ length improves the system reliability, these come with higher overheads. Next, we report the recovery overheads within 20,000 seconds simulation time where Table 4.19 records the recovery messages per second, M/T and Table 4.20 reports the data transferred per second, L/t . The transferred data refers to the total size of bias elements and the total size of keys (identifiers) that are exchanged among peers during recovery process^{4.11}. We found that the recovery overhead is highest when δ is 100 seconds and r is 5, while it is smallest when δ is 800 and $r = 2$. However, if we look back at Figure 4.24, it is clear that the high reliability of the P2P-GN can be achieved when $\delta = 100$ seconds and $r = 5$ albeit at the cost of high recovery overheads. While, when $\delta = 800$ seconds and $r = 2$, the system loses bias elements significantly in a very short period. This reflects the need of consideration on the trade-off between the reliability and recovery overheads in determining the value of δ and r .

^{4.11}Note that we measure this by obtaining the size of the storage structure (HashMap in our Java implementation) that holds the transferred bias elements and keys. Therefore, the additional message overheads were not included.

Table 4.19: The recovery messages per second (M/t) within 20,000 seconds of simulation time.

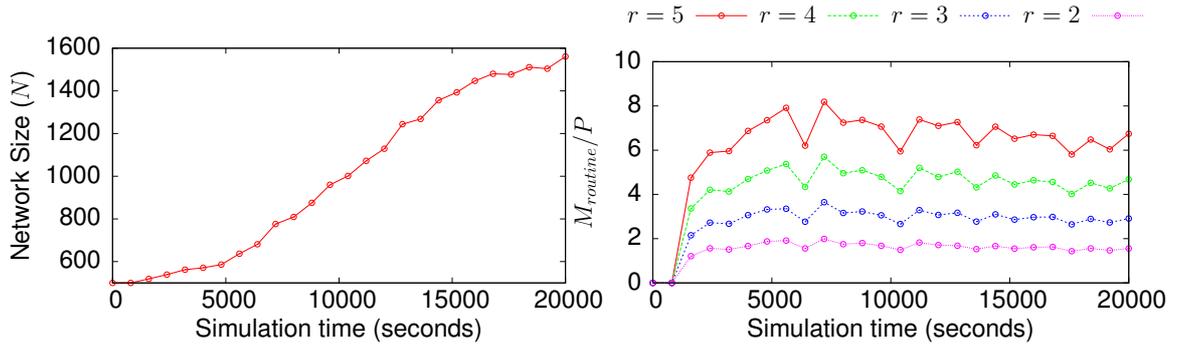
r	M/t			
	$\delta = 100$	$\delta = 200$	$\delta = 400$	$\delta = 800$
2	3.20	3.06	2.82	2.41
3	5.30	5.09	4.77	4.22
4	7.88	7.56	7.12	6.43
5	10.94	10.54	10.00	9.17

Table 4.20: The data transferred per second (L/t) within 20,000 seconds of simulation time.

r	$L/t(\text{in Mb})$			
	$\delta = 100$	$\delta = 200$	$\delta = 400$	$\delta = 800$
2	0.0982	0.0928	0.0835	0.0686
3	0.1630	0.1553	0.1436	0.1243
4	0.2424	0.2309	0.2149	0.1914
5	0.3366	0.3222	0.3023	0.2728

The amount of data transferred (L/t) for recovery is relatively small compared to the total storage of master bias elements that is 23.28 Mb (see Table 4.20). This proves that our fault-tolerance scheme is efficient although operating within a highly chaos network.

Next, we specifically measured the number of recovery messages per peer during routine recovery session $M_{routine}/P$ as shown in Figure 4.26. Figure 4.26a shows the increase in network size (N) during the simulation time. We started the network from 500 peers and it grew to 1,561 peers in 20,000 seconds of simulation time. The $M_{routine}/P$ that is reported in Figure 4.26b exhibits constant growth in simulation time. This effectively suggests that the number of recovery message is independent to the network size (since the network size is growing in time in this simulation). Hence, our fault-tolerance approach is scalable for large networks. In addition, we found that there is no communication at all during recovery session when there is no leaving or joining peers at all in the network—at less than or equal to 1,000 seconds of simulation time (see Figure 4.26b). This demonstrates that the restrictions on



(a) Network size (N) by simulation time. (b) The number of recovery messages per peer in every `routineRecovery(.)` execution.

Figure 4.26: The growth in N (network size) and $M_{routine}/P$ (the number of messages per peer in every `routineRecovery(.)` execution) during routine recovery session with recovery $r = \{2, 3, 4, 5\}$ and $\delta = 800$ under churn with $MTTF = 5,000$ seconds within 20,000 seconds of simulation time.

communication during routine recovery by condition C_{pre} and C_{suc} (as given in Equations 4.16 and 4.22, respectively) ensure that our fault tolerance scheme is efficient as it only performs recovery when it is needed.

4.8 Summary

In this chapter, we have presented an efficient computing approach for pattern recognition within a server-less P2P system in dealing with large datasets. Our work suits the naturally distributed environment well^{4.12} and can provide an efficient alternative for scalable pattern classification at a low cost. In this chapter, we have also presented a fault-tolerance management to support the P2P-GN implementation within dynamic networks. The distributed nature of our scheme provides a degree of fault tolerance to the system where the proposed algorithm is able to provide a correct result, despite facing a high churn. This scheme deals with the dynamic networks efficiently through a reactive routine recovery protocol, where the recovery is only performed under restrictions to avoid unnecessary communications. Additionally, the Chord overlay network handles the leaving and failing nodes automatically

^{4.12}A geographically distributed environment where each site collects its own data.

and maintains the link between peers efficiently. In summary, the key findings in this chapter are described as follows:

- In the comparative analysis with the BPNN, HGN and DHGN, we have shown that the communication overhead in the P2P-GN (peer level and network level) is significantly low compared to other methods.
- The convergecast recall improves the efficiency of the P2P-GN in dealing with high-dimensional problems. It requires a constant communication overhead for each peer, therefore, the algorithm becomes very scalable for large-scale distributed networks.
- Our experiment shows that the distributed P2P-GN approach reduces the run-time and storage overheads per host in the centralised, state-of-the-art classifiers (BPNN, RBFN, 1-NN, k -NN and ID3). The run-time and storage overheads per host in the P2P-GN remain low for large datasets. This demonstrates that the P2P-GN provides an economical option for scalable classification algorithm of large datasets through a fully-distributed approach.
- We have experimentally and theoretically demonstrated that the local learning time at leaf nodes in the P2P-GN grows at a constant rate with an increase in the number of stored instances and with an increase in data dimension. This suggests that the P2P-GN can perform an online learning efficiently and thus suitable to process large datasets and a high volume of data streams.
- Unlike in the flat recall, the recall time in the convergecast recall remains low with an increase in data dimension. This further demonstrate the efficiency of the convergecast recall for high-dimensional problems.
- There is a magnitude of conflicts in the flat recall which with the growth in stored training instances, results in a small increase in computational time. Although this time increase is negligible, as it grows on a very low scale, this issues is worth investigating in future work.

- The proposed fault tolerance scheme is proven effective to sustain the P2P-GN correctness throughout its lifetime despite operating under a very high churn (Weibull churn model with $MTTF = 5,000$ seconds). The master bias element availability remains optimal at 1 within simulation time of 20,000 seconds. This, however, is subject to the correct selection of replication rate r and recovery interval δ . A longer δ length requires more replicas (higher r) to keep operating correctly, despite losing a high number of replicas.
- The number of replicas across the network is maintained at r and this shows that our fault-tolerance scheme is storage-efficient, as it avoids the wasting of storage for unnecessary and expired replicas.
- The proposed fault tolerance scheme is a local and reactive protocol, where a peer only performs replication recovery when there are any changes to its closest neighbours. Hence, this is an effective and efficient fault tolerance scheme for such fully-distributed systems. The simulation results show that the data transferred per second is only approximately 1% of the total storage size across the network and the communication overhead for recovery is independent of the network size. Zero communication is required when the network is in a stable state.

The convergecast P2P-GN solves high-dimensional data problems by associating processors where each processor recognises a subset of features and the recall results from these processors are then combined into a global recall. The memory is stored and processed in parallel by multiple different nodes at different sites.

Our experiment results show that the P2P-GN can provide a scalable, resource-efficient and online learning algorithm through a fully-distributed approach. This has been demonstrated in the experiment using two large datasets (Waveform-40 and LED). The k -NN and 1-NN also perform online learning, however, their storage and recall time are significantly higher compared to the P2P-GN.

The P2P-GN maintains a low execution time complexity per peer and this resource-efficiency characteristic is appealing in a serverless-distributed computing environment, where the majority of participating peers are low performance machines. The local recall time at a requester in the flat recall method is affected by the number of stored instances in a very small scale where it increases slowly with an increase in the stored instances, but the local recall time at other nodes remain constant. The local recall time at a requester in the flat recall is however, linearly dependent on the data dimension. In the convergecast approach, the local recall time grows at a constant rate with an increase in the number of stored instances and with an increase in data dimension. This shows the efficiency of our scheme to perform an online learning and efficient computation for large and high-dimensional datasets.

Our method performs a single-cycle operation, and hence the amount of communication among peers is low during both learning and classification phases in comparison to other learning schemes, as iterative learning is avoided. Given a high potential of frequent data updates within a network, the online learning enables cheap per-item learning, without rebuilding the model, which is usually costly in terms of communication and CPU processing. Clearly, the proposed method deals efficiently with large data. Thus, it is worth consideration as an economical solution for large datasets classification problems especially when dealing with in the absence of a high-performance server.

Chapter 5

Distributed Spam Detection using DASMET

Spam has been described in previous studies [115, 113, 40] as one of the major problems in P2P applications. Spam appears in various forms. In email and messenger applications, spammers also use images to make such spam harder to detect, than the more common text spam. In messenger applications, spam may appear in the form of fake contacts which continuously send advertisements. Sometimes, spammers even use real contacts accounts to send messages. It is notoriously used as a tool to control copyright infringement in P2P systems by the content owner, where it appears in the form of malware or polluted content which floods the system with bogus contents. Recently, audio and video P2P streaming applications are also spammed with corrupted streaming chunks, which alter or disrupt the streaming service. Given that spam has appeared in a variety of forms, a spam filter requires significant efforts to learn the complex patterns in order to effectively classify these new forms.

The popular distributed spam detection methods within P2P networks are usually reputation-based approaches [52, 198, 201, 202]. However, most of these methods cannot classify a “never-been-seen” or fuzzy item. The rating or the report based on users’ experiences of the item, are required.

Spam is often generated by machines using a common template and are sent in bulk to many users. Hence, these forms of spam are usually similar or exact duplicates. The duplicate spam items can be easily filtered out using a signature-based approach, where the signatures of the spam items are stored in a centralised database. This, however is infeasible for fully-distributed and large networks like P2P networks. Moreover, the signature-based methods are also unable to recognise the fuzzy spam items and therefore, a degree of noise is often inserted by spammers to avoid the detection using signature-based methods. Zhou et al. [220] proposed a distributed spam detection system that uses feature signatures to identify spam emails and it can recognise fuzzy and “never-been-seen” spam items. Nonetheless, the reputation-based methods and the method in Zhou et al. [220] require the unique identifiers of all identified spam objects to be stored. This results in a high storage overhead since a botnet (a program to automatically generate spam) can easily generate a high volume of spam objects. Moreover, in the reputation-based approaches [52, 202, 198], the high volume of spam objects results a high communication overhead since votes for each object are exchanged among peers; and in Zhou et al. [220], the identifiers for all objects which have one or more features that match with features of a queried object (test instance) are sent to the requester peer (the peer which requests for classification).

Pattern recognition techniques have been proven effective for spam detections [161, 127] and these do not require the storing of the identifiers of all identified spam items as in Zhou et al. [220]. However, most of the available pattern recognition algorithms are centralised. Considering that P2P networks work without a server, the available pattern recognition approaches are infeasible for P2P networks, since these require a fully-distributed approach. Therefore, this chapter investigates an efficient (time and communication), fully-distributed pattern recognition algorithm to detect spam within a P2P network.

Initially, we explored the effectiveness and efficiency of the P2P-GN for the spam detection problems. Since such problems involve a large number of duplicate or

near-duplicate (almost similar) objects [131], we suggest that a more efficient (time and communication) solution can be achieved in classifying these objects. This is accomplished by extending the flat structure of the P2P-GN into a tree structure distributed associative memory. In the P2P-GN, the fine-grained memories of trained patterns are stored at leaf nodes and the association of these memories enables it to recognise fuzzy patterns. We extend this basic P2P-GN functionality by storing the memories in several different levels of view—in a tree structure where a node at the higher level of the tree stores a larger patterns’ view (more coarse-grained memories) and the root node stores the whole patterns’ view. Hence, the duplicate patterns can be detected in a straightforward way by the root, while the noisy patterns can be recalled by the nodes at the lower part of the tree (which store fine-grained memories).

This comes with a trade-off of more storage overhead compared to the P2P-GN. Nonetheless, this overhead can still be considered as low since k training objects (instances) with exactly similar features (duplicate patterns) are only stored once—these are considered as a single unique pattern. Therefore, the trade-off can be ignored here; instead, we focus on improving the response time and communication overhead. The resulting extension of the P2P-GN is called the Distributed Associative Memory Tree (DASMET). In this chapter, we study the feasibility of the DASMET to provide a fast, communication-efficient and fully-distributed solution for the spam problems.

Two spam problems have been identified for evaluating our approach: email spam and image spam. Since email spam is common, we include it here as a practical application of our approach. Alternatively, image spam detection provides sophisticated content-based detection that is useful, since the current trend in spamming within P2P system is moving towards multimedia spam objects.

This chapter consists of six sections as follows. The inefficiency of the P2P-GN approach for the classification problems with a high number of repeating patterns

which led to the development of the DASMET for spam detections are briefly explained in Section 5.1. Then, we describe our proposed spam detection system in Section 5.2. In Section 5.3, we introduce the DASMET and describe its learning and recall^{5.1} operations. Following this, we report the outcome of our experimental study on the DASMET’s effectiveness and efficiency for distributed email spam detection and distributed image spam detection in Section 5.4. Next, in Section 5.5, we theoretically analyse the complexity of the proposed scheme. Finally, we summarise the findings of this chapter in Section 5.6. For reference, the notations that are used in this chapter are summarised in Table A.3 of Appendix A.

5.1 Efficient Classification for Datasets with Frequent Repeating Patterns

In the P2P-GN, the memory of a pattern \mathbf{x} is divided into n_H small parts and these parts are stored at n_H leaf nodes. To recall the memory of \mathbf{x} , a requester sends recall queries to all n_H leaf nodes and these leaf nodes response with their recall predictions. The recall operation of a clean copy of a stored pattern \mathbf{x} is similar to the recall of a noisy copy of \mathbf{x} . Hence, the recall process for both the clean and noisy copy of the stored \mathbf{x} incur similar run-time and communication overhead. This results in inefficiency in the recall of duplicates of the stored patterns.

Let say the recall requests for a duplicate of x occurs at z times, then the total number of messages equals $2n_H \cdot z$. The number of messages can be reduced to $2z$ by storing the whole memory of x at a single node p_a where a recall request for a duplicate of x is sent to a single node p_a instead of n_H leaf nodes. Meanwhile, a fuzzy pattern can still be predicted by recalling the memories at leaf nodes. This idea leads to the development of an extension of the P2P-GN for datasets with a high amount of repeating and similar patterns, that is the DASMET.

^{5.1}Recall refers to classification or prediction in this context

DASMET stores patterns in a tree structure; the leaf nodes store the fine-grained memory, and the nodes at upper layer store a more coarse-grained memory where the top node has an overall view of the pattern. To recall a pattern, the algorithm initially detects the pattern at the root. If the pattern has been found, then the recall process is stopped here. Otherwise, we continue to look at a more close-up (fine-grained) view of the pattern at the lower-level nodes recursively until we discover the most likely predictions. By doing so, we can reduce the communication and processing overheads by detecting duplicates or closely-resembling patterns at the entry point of the system and avoid wasting resources on an expensive pattern recognition process. This is achievable because a high number of exact duplicate and similar patterns in the system is likely.

The brief idea behind our method is shown in Figure 5.1. The example shows

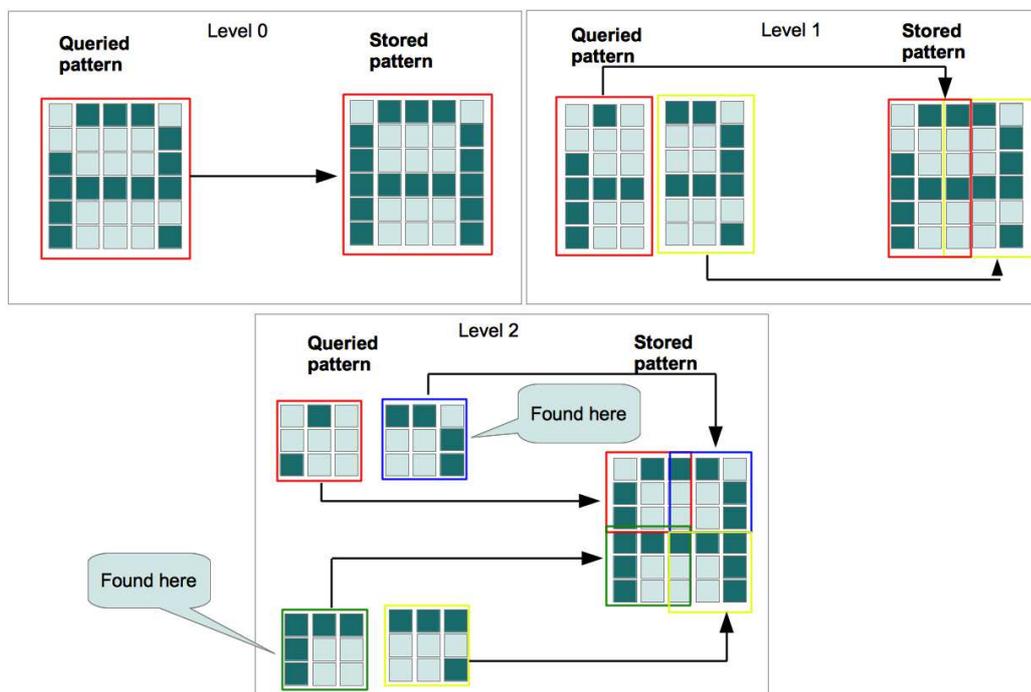


Figure 5.1: An example of the idea used in the spam detection using DASMET. In level 0, the whole queried pattern is matched with the overall stored pattern and if it is not found, then in the second level, two parts of the pattern are matched against the stored pattern. The coloured lines shows the matched pattern—for instance, the blue region of the queried pattern is matched with the blue region of the stored pattern. In this example, the similar patterns are found for two parts out of four at level 2.

that the queried pattern is first matched with the stored pattern at level 0; if it is found to match the stored pattern, then the detection process decides that the label of the queried pattern equals the matched stored pattern. If it is not found, then the queried pattern is split into two smaller parts and these parts are matched with the equivalent parts of the stored pattern at level 1. For example, the region specified by the red line in the queried pattern is matched with the region covered by the red line of the stored pattern. This process is continued until one of the matched region are found; this requires the memory of the whole segment and its parts. Instead of storing the pattern in its raw form, we only store the key of the pattern (pattern identifier). The number of iterations (levels) is also limited by the height of the tree since the smallest sub-pattern size is predefined (see Rule 5.1 of Section 5.3.1).

5.2 Distributed Spam Detection System

The functional components of the proposed distributed spam detection system are described in Figure 5.2. The system has two view levels: peer view and network view. The peer view involves the components that are used by the requester, while the network view involves the components that perform in-network processing, involving many other peers within the network. The peer view consists of five components, as follows.

- feature extraction: extracts features using an extraction tool and forms a feature vector which summarises the querying object.
- pre-processing: performs discretisation on the feature vectors which form a pattern.
- bias identifier generation: generates the pattern's identifiers from the processed feature vector based on the abstract DASMET tree.
- request submission: prepares the learning or recall queries for the generated bias identifiers. These queries are then sent for in-network processing.

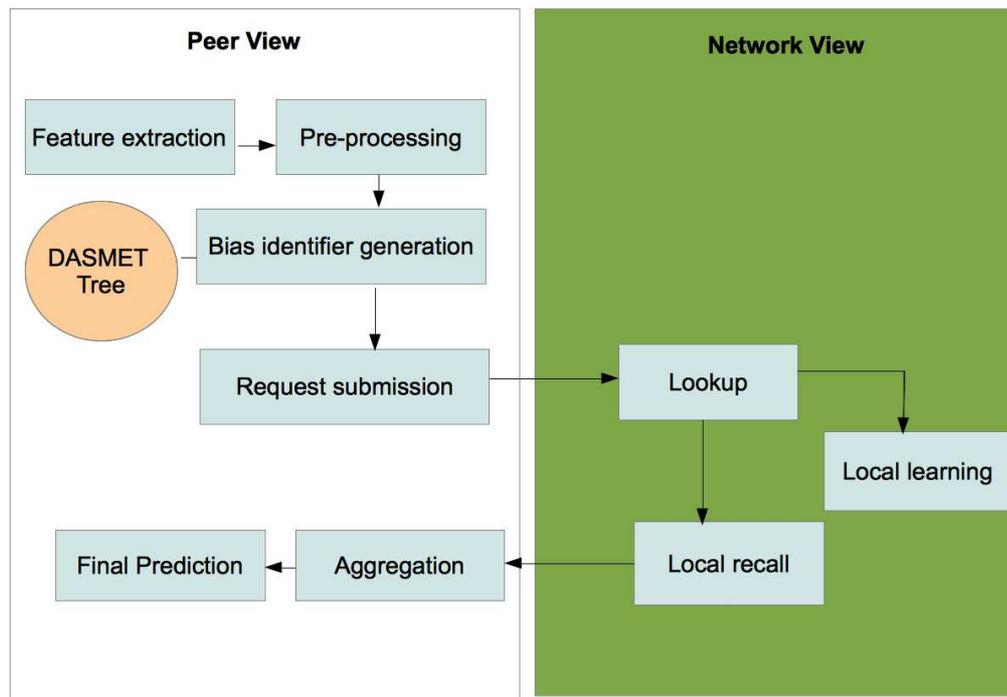


Figure 5.2: Functional components of the distributed spam detection system.

- aggregation: receives the local predictions from the local recall component within the network and performs aggregation using these local predictions. Then, it sends the aggregated prediction to the final prediction component.
- final prediction: makes a final decision making using the aggregation prediction from the aggregation component.

The network view encompasses three components as follows.

- lookup: find the location of the querying bias identifiers.
- local learning: involves a group of peers which perform learning independently at their site and process the incoming learn requests simultaneously.
- local recall: involves a group of peers which perform recall independently at their site. They work in parallel to process the incoming recall requests and then send the local predictions to the requester.

During learning, the requester peer extracts features from the item of interest and performs pre-processing on the extracted features to form a feature vector. Using

the template of the abstract DASMET tree which has been constructed when the peer joined the system for the first time, a set of identifiers $\hat{\psi} = \{\psi_1, \psi_2, \dots, \psi_k\}$ are generated from the feature vector. An identifier and the associated label (class of the feature vector) forms a pair $\langle \psi, label \rangle$. Hence, a set of pairs $\{\langle \psi_i, label \rangle\}_{i=1}^k$ are created from the generated identifiers. These pairs are then sent to the network using a lookup process and the peers that are responsible for the identifiers then learn the pairs locally. The feature extraction, pre-processing and identifier generation during recall process are basically similar to the learning process, except that the recall queries do not include the associated label (since that a recall query is sent to find the associated label). The local recall is performed in parallel at a number of peers and the local results from these peers are sent back to the requester, where the aggregation and final prediction are executed.

5.3 DASMET for Pattern Recognition within P2P Networks

The DASMET structure is presented next. It constructs the logical structure of the distributed algorithms, where every node in the tree represents a computational unit. Every peer within the network builds their own abstract DASMET tree structure using the parameters that are provided upon joining the system. As a result, all peers construct the equivalent trees. The DASMET tree is used for reference in generating identifiers throughout the DASMET scheme's lifetime.

5.3.1 The Logical Structure of The DASMET

The logical structure of the DASMET is a rooted tree $D_R = (V_R, E_R)$, where V_R is the set of vertices and E_R is the set of edges (see Figure 5.3). Each vertex in DASMET is a logical storage unit, v_i (that is $v_i \in V_R$); it logically stores a set of bias elements which include score tables. The distributed storage of the DASMET is the same as in the P2P-GN (as described in Section 3.1.3 of Chapter 5). The storage

structure for nodes at level- \hbar (leaf nodes) is exactly the same in the P2P-GN, but with additional memories stored for the upper level of the tree (internal nodes and a root node). Unlike in the P2P-GN where a requester node acts as the root node, a requester node is an internal entity to the DASMET. A requester in both the P2P-GN and the DASMET does not store any memory, instead, it only requests for learning process or recall process and performs aggregations in calculating prediction for its own request. A requester node can be any peer within P2P networks.

The DASMET structure depends on two predefined values: (i) the maximum number of children of each node, or the segment size at a leaf node, φ_s ; and (ii) the number of features d . The DASMET operation starts by creating a number of sub-patterns $\widehat{X} = \{\hat{x}_i\}_{i=1}^{n_H}$ prior building the tree. The process of creating the sub-patterns is similar to that of the P2P-GN as described in Section 3.1 of Chapter 3. Using the input \widehat{X} , the DASMET tree is generated by Algorithm C.1 of Appendix C. Note that m in this algorithm is equal to φ_s .

Figure 5.3 shows an example of DASMET structure for a problem with the number of features is 23. \mathcal{SF} is segmented into 10 segments of size five each. The leaf nodes stores the sub-patterns, while, the internal nodes and the root node store the combined indices from its children. The number of leaf nodes, n_H , is equal to Equation 3.1 of Chapter 3. The tree height, \hbar and the total number of nodes, n_R are formally given in following Equation 5.1 and Equation 5.2, respectively.

$$(\log_{\varphi_s} n_H + 1) > \hbar \geq \log_{\varphi_s} n_H \quad (5.1)$$

$$\begin{aligned} n_R &\leq \sum_{l=0}^{\hbar-1} \varphi_s^l + n_H \\ &\leq \frac{\varphi_s^{\hbar}-1}{\varphi_s-1} + n_H \end{aligned} \quad (5.2)$$

Each leaf node h_i memorises a sub-pattern \hat{x}_i which is a sequence of values that represent \widehat{sf}_i and then the parent node stores the association of memories from its children. At the end, the root node of the tree stores the associative memory of the

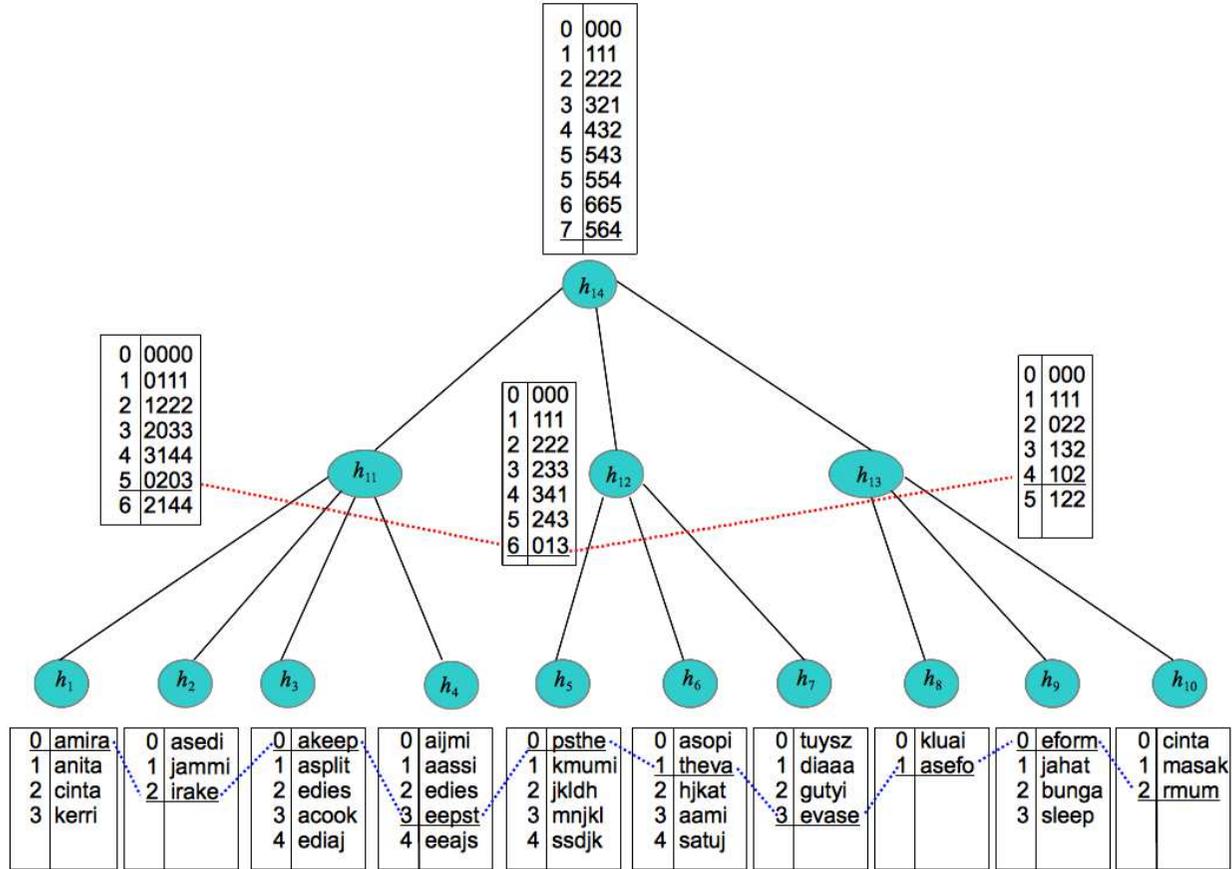


Figure 5.3: An example of a DASMET architecture for a problem with d equals to 23 and a newly added input pattern “*amirakeepsthevaseformum*”. The predefined φ_s is five and therefore, the input pattern is segmented into 10 sub-patterns and each sub-pattern is assigned to a leaf node, respectively. It has 10 leaf nodes, two internal nodes and one root node with the height of the tree is two. Each node holds a bias array (represented by the box) which stores the memory of the learnt pattern. The underlined entries represent the active entries for the input pattern “*amirakeepsthevaseformum*” and the dotted lines show the links of the memories.

whole pattern. The memories stored at a parent node strengthen its memory since it has more memory associations than its children.

A memory of a unique pattern at a root node can be recalled by linking the entries within bias arrays at any level l . As shown in Figure 5.3, the pattern “ami-rakeepsthevaseformum” can be recalled at any level by associating entries from nodes at the same level (see the dotted lines linking the entries from the bias arrays) as follows.

- level 0 - by the node h_{14} at entry 7.
- level 1 - by associating the entries: entry 5 at node h_{11} and entry 6 at node h_{12} and entry 4 at node h_{13} .
- level 2 - by associating the entries: entry 0 at node h_1 , entry 2 at node h_2 , entry 0 at node h_3 , entry 3 at node h_4 , entry 0 at node h_5 , entry 1 at node h_6 , entry 3 at node h_7 , entry 1 at node h_8 , entry 0 at node h_9 , and entry 2 at node h_{10} .

The selection of φ_s 's value affects the performance of the algorithm and therefore, a cross-validation may be required to make the decision. A large φ_s may reduce the accuracy of the DASMET by degrading the system's ability to recognise a highly fuzzy pattern. On the other hand, a small φ_s leads in an increase in the communication overhead and may result in the worst-case of recall run-time.

A smaller φ_s gives a greater number of sub-patterns (n_H) and total nodes (n_R). Thus, the parent peer that is responsible for aggregating the result, will be burdened with too many received transactions to process. Consequently, the growth in n_H increases the height of the tree (as shown in Equation 5.1) which later increases the communication delay and effectively increases the run-time. Meanwhile, an increase in n_R increases the overall communication overheads.

A large φ_s , on the other hand, means each sub-pattern stores a large features association and thus this may cause the scheme to lose its precision when dealing with a highly distorted pattern. These highly distorted patterns can be explained

as follows. Consider that the sub-patterns of $\mathbf{x} = \text{“}amirakeepsthevaseformum\text{”}$ at h_2, h_4, h_6, h_8 and h_{10} from Figure 5.3 are distorted. Thus, \mathbf{x} is undetected at level 1 (by associating h_{11} and h_{12}) and at level 0 (h_{13}). However, the recognition can still be made at level 2 by joining memories from all ten nodes at level 2 ($\{h_1, h_2, h_3, h_4, h_5, h_6, h_7, h_8, h_9, h_{10}\}$). If φ_s in this example is set to 10 instead of five, the prediction cannot be made since all sub-patterns at leaf nodes are distorted.

For simplicity, we define a rule-of-thumb on φ_s and OV which will be used throughout this thesis as follows.

Rule 5.1 (DASMET Rule-of-Thumb). *Given a pattern P with size- d , the value of φ_s is defined to be $2 \leq \varphi_s \leq \lfloor \log_2 d \rfloor$ with OV equal to $\lceil \frac{\varphi_s}{2} \rceil$ where $d > \varphi_s$.*

By limiting the maximum φ_s to $\lfloor \log_2 d \rfloor$, we can ensure the DASMET’s capability to recognise a noisy pattern, while, the minimum value of φ_s limits the communication overhead.

5.3.2 Network-wide DASMET

In the implementation within P2P networks, the bias array is divided into fine-granularity components where each component includes a score table. This forms a bias element which is similar to the storage in the P2P-GN (see Definition 3.4 and Definition 3.5 of Chapter 3). A bias identifier provides an address to every bias element across the network. The process to generate the bias identifier is explained as follows.

5.3.3 Bias Identifier Generation

Figure 5.4 shows an example of sub-patterns and bias identifiers created from the pattern *“amirakeepsthevaseformum”*. In this example, 10 sub-patterns with size five each are created. The sub-patterns and their bias identifiers (see Definition 3.6 of Chapter 3) are shown in the form of pair $\langle \text{bias identifier} : \text{input} \rangle$ ^{5.2}. There

^{5.2}The input for generating a bias identifier at a leaf node is a raw sub-pattern, while, the input at the internal and rootnode is a sequence of combined identifiers of its child nodes.

are 14 bias identifiers in total where 10 bias identifiers represent the sub-patterns at the level 2; three bias identifiers represent the sub-patterns at level 1; and one bias identifier represents the whole pattern at the level 0. If all of these bias identifiers have not been seen yet in the system (in case that this is the first time the pattern “*amirakeepsthevaseformum*” occurs), then the learning process of these bias identifiers creates 14 new bias elements within the system (a bias identifier for each bias element).

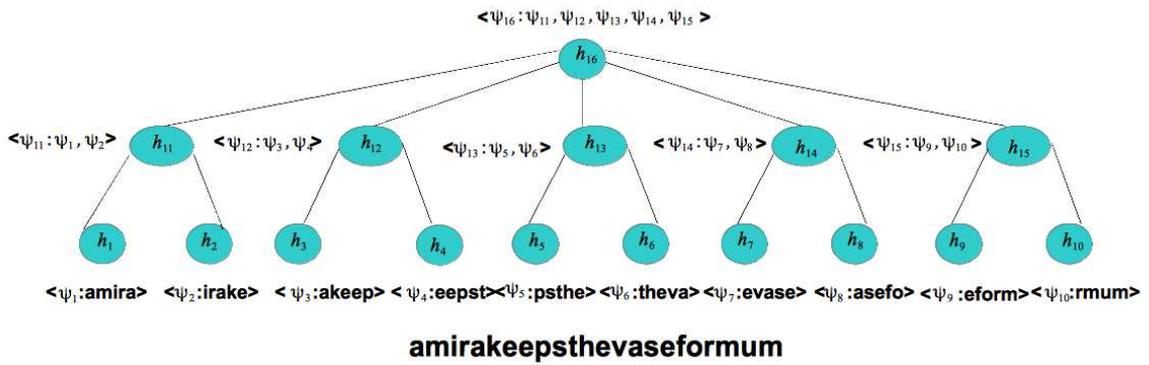


Figure 5.4: Examples of pairs $\langle \text{bias identifier:input} \rangle$ that are created for pattern “*amirakeepsthevaseformum*”. In total there are 14 bias identifiers $\{\psi_1, \psi_2, \dots, \psi_{14}\}$ since there are 10 identifiers at level 2, three identifiers at level 1 and an identifier at level 0. In case that these 16 bias identifiers have not been stored yet in the system, then 16 bias elements are created from these bias identifiers.

The procedure for generating bias identifiers are described in Algorithm 5.1. The bias identifiers for every node h_i in G_R are calculated from input \mathbf{x} starting from level \hat{h} until it reaches level 0. The generation of bias identifiers at leaf nodes (level \hat{h}) in the DASMET is similar to that of the P2P-GN in Chapter 3 where the input to the identifier generation function $\text{qLeaf}(\cdot)$ is the sub-pattern and the position or index of the leaf node. However, an input argument to the identifier generation function $\text{q}(\cdot)$, for an entry at an internal/root node v , is an ordered set of combined identifiers from its child nodes $\hat{\psi} = \{\psi_1, \psi_2, \dots, \psi_{w_v}\}$, where w_v is the number of child nodes and ψ_i is a bias identifier from i th child of v . Here, a hash function is used for the function $\text{q}(\cdot)$ and $\text{qLeaf}(\cdot)$.

Algorithm 5.1. generatingIdentifier(\mathbf{x})

```

1:  $l \leftarrow \bar{h}$ 
2: Create segments to learn,  $\{\hat{x}_1, \hat{x}_2, \dots, \hat{x}_{n_H}\}$  from  $\mathbf{x}$ 
3:  $i \leftarrow 1$ 
4: while  $i \leq n_H$  do
5:   Calculate bias identifier for a leaf node  $h_i$ ,  $\psi(h_i) \leftarrow \mathbf{qLeaf}(\hat{x}_i, i)$ .
6:    $i \leftarrow i + 1$ 
7: end while
8:  $l \leftarrow l - 1$ 
9: while  $l \geq 0$  do
10:   $i \leftarrow 1$ 
11:   $m_l \leftarrow$  number of nodes at level- $l$ .
12:  while  $i \leq m_l$  do
13:     $\{\psi_{(j,i)}\}_{j=1}^{w_i}$  is a set of bias identifiers of  $w_i$  child nodes of internal/root node  $i$ .
14:    Calculate bias identifier for internal/root node  $h_i$ ,  $\psi(h_i) \leftarrow \mathbf{q}(\psi_{(j,i)})_{j=1}^{w_i}$ 
15:     $i \leftarrow i + 1$ 
16:  end while
17:   $l \leftarrow l - 1$ 
18: end while

```

Definition 5.2 (Function $\mathbf{qLeaf}(s, j)$). *An identifier generation function $\mathbf{qLeaf}(s, j)$ at a leaf node creates a bias identifier for a given input pattern s and position j .*

Definition 5.3 (Function $\mathbf{q}(\hat{\psi})$). *An identifier generation function $\mathbf{q}(\hat{\psi})$ at an internal or root node creates a bias identifier for a given input of bias identifiers of its $|w|$ children $\hat{\psi} = \{\psi_1, \psi_2, \dots, \psi_{|w|}\}$.*

Any host is capable of calculating identifiers of a particular vector \mathbf{x} by itself, provided that the G_R structure is known to all peers. Given the common parameters d and φ_s , a peer knows the G_R structure by building its abstract structure locally. This structure is used as a framework to create all bias identifiers of pattern \mathbf{x} .

In an example in Figure 5.5, four bias elements and identifiers are created at the root node (i.e., $\{\psi_{30}, \psi_{27}, \psi_{20}, \psi_{16}\}$). The bias identifiers at the root node are created using the combined identifiers from the leaf nodes {leaf node 1, leaf node 2, leaf node 3}. For instance, the bias identifier ψ_{30} (at the root node) is created using the combination of identifiers ψ_3 (in the leaf node 1), ψ_8 (in the leaf node 2) and ψ_{11} (in the leaf node 3) as input parameters, while, a bias identifier for a bias element at a leaf node is generated by using an input sub-pattern.

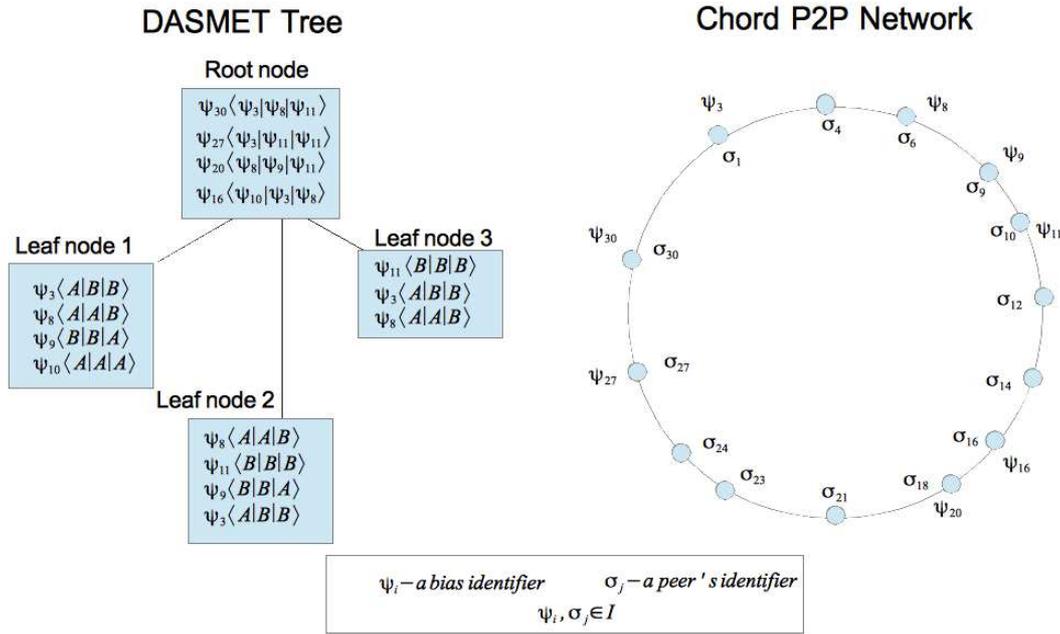


Figure 5.5: The diagram on left shows the logical location of bias arrays and bias elements within DASMET structure, while, the diagram on right shows the placement of bias elements within a Chord network. Bias identifiers are generated in the same identifier space as peers' identifiers. A bias element with identifier ψ_8 being assigned to the peer σ_6 where the peer σ_6 is responsible to store bias elements with identifiers within the range $[6, 9)$.

In the context of P2P networks, an identifier or a key of a data must comply with the overlay network identifier space for efficient storing and lookup. The consistent hash function that creates the identifier is defined by the P2P overlay design and it is provided to the peer when it joins the network. In our implementation, we use SHA-1 to create the identifiers. In the placement within P2P networks as shown in Figure 5.5, a bias element with identifier ψ is placed at a peer which has identifier $p.ID$ where the ψ is within the range of $[p.ID, p_{suc[0].ID})$, $p.ID$ is an identifier of p and $p_{suc[0].ID}$ is an identifier of the immediate successor of p .

Here, we use only sub-patterns as input for bias identifier generation. For instance, the leaf node 1, leaf node 2 and leaf node 3 have similar sub-patterns $\langle A|A|B \rangle$ and therefore, the same identifiers ψ_8 are generated for these sub-patterns. However, henceforth in this thesis, we use the pair of sub-pattern and position as input for bias identifier generation. For a pattern with the same domain values (e.g., $\{0, 1\}$),

there is a high possibility that a segment \hat{x}_i at the position i is similar to a segment \hat{s}_j at the position j . Therefore, the use of only sub-patterns for identifier generation will result in a smaller number of bias elements compared to the use of the pairs of sub-pattern and position. However, this may also result in the peer which holds the bias elements responding to a greater number of recall or learning requests.

5.3.4 DASMET Learning Procedure

The learning process in the DASMET strategy is explained in Algorithm 5.2.

Algorithm 5.2. DASMET Learning

```

1: Input :  $c$ ,  $\mathbf{x}$ , and  $H$ 
2: At requester peer  $\mathbf{x}$ 
3:  $\{\psi(h_1), \psi(h_2), \dots, \psi(h_{n_R})\} \leftarrow \text{generatingIdentifier}(l, \mathbf{x})$ .
4: for all  $h_i \in G_R$  do
5:   Sends LEARN_REQUEST for  $\{\langle \psi(h_i), c \rangle$  to the network.
6: end for
7:
8: At a peer  $\mathbf{y}$ : upon receiving LEARN_REQUEST from peer  $\mathbf{x}$ 
9: Perform localLearning(.).

```

The input to the DASMET learning is a set of leaf nodes H , and a training instance which consists of a label for the training instance c and a sequence of feature vectors \mathbf{x} (which forms a pattern). In the DASMET learning, all the indices are calculated at a requester peer (see Section 5.3.3). In this case, all peers are assumed to know the tree structures (which they can built themselves locally given the tree parameters) and the identifier generation function. This information can be shared during peers' bootstrapping. Therefore, all peers know \bar{h} , that is, the height of the DASMET tree. The `localLearning(.)` function and `LEARN_REQUEST` message here are equivalent to the P2P-GN in Chapter 3.

The bias identifiers for all nodes in DASMET (G_R) and class label for the training instance in pairs $\{\langle \psi(h_i), c \rangle\}_{h_i \in G_R}$ are sent to the network using the `LEARN_REQUEST` messages where $\psi(h_i)$ is the bias identifier that is generated for node $h_i \in G_R$ and c is the class label. These bias identifiers are generated using the `generatingIdentifiers(.)` function which has been discussed in the previous Section 5.3.3. Upon receiving the

LEARN_REQUEST message, recipient peers then locally perform the local learning and the learning process (see Section 3.1.7 of Chapter 3).

5.3.5 DASMET Recall Procedure

The procedure during recall is explained in the Algorithm 5.3 and an illustrative example of recall process is given in Section D.5 of Appendix D. Three main functions—

Algorithm 5.3. DASMET Recall

```

1: At requester peer x
2:  $\{\psi(h_1), \psi(h_2), \dots, \psi(h_{n_R})\} \leftarrow \text{generatingIdentifier}(l, \mathbf{x})$ .
3:  $l \leftarrow 0$ 
4:  $h_1 \leftarrow$  root node.
5: Sends RECALL_REQUEST for  $\psi(h_1)$ .
6:
7: At a peer z: upon receiving RECALL_REQUEST  $\psi(h)$  from a peer x
8: Calculate prediction  $\hat{r}(h)$  using function  $\text{localRecall}(\psi(h))$ .
9: Send RECALL_RESPONSE with  $\hat{r}(h)$  to the peer  $x$ .
10:
11: At requester peer x: upon receiving responses  $\hat{r}(h)$ 
12: if  $l \neq 0$  then
13:    $w_x \leftarrow$  the number of children of the peer  $x$ .
14:   Aggregate the predictions to obtain a prediction  $\hat{R}(G)$  using  $\text{agg}(\{\hat{r}(h)_i\}_{i=0}^{w_x})$ 
15: else
16:    $\hat{R}(G) = \hat{r}(h)$ 
17: end if
18: if  $C_0 == \text{true}$  then
19:    $\mathcal{L} = \text{finalPrediction}(\hat{R}(G))$ .
20: else
21:    $l ++$ 
22:    $H_l \leftarrow$  is a set of nodes at level- $l$ .
23:   for all  $h_i \in H_l$  do
24:     Sends RECALL_REQUEST for  $\psi(h_i)$ .
25:   end for
26: end if

```

$\text{localRecall}(\cdot)$, $\text{agg}(\cdot)$, and $\text{finalPrediction}(\cdot)$ —and two types of messages—RECALL_REQUEST and RECALL_RESPONSE—which are used here, are the same functions as in the P2P-GN in Chapter 3.

The recall is invoked by a requester peer and it starts from the root node at level $l = 0$ and after completing the $\text{generatingIdentifier}(\cdot)$ execution. The lookup begins by sending a query RECALL_REQUEST for identifier $\{\psi(h_1)\}$ to the network. If the receiving peer has the queried index, then it calculates the prediction using the local

function `localRecall`($\psi(h_1)$) and then responds with the `RECALL_RESPONSE` message with its local prediction $r(h_1)$. Upon receiving the response, the requester validates the prediction using Rule 5.5 and Rule 5.4.

Rule 5.4 (Termination Condition C_0). *The termination condition, C_0 of this algorithm is true in the event of: (i) a single prediction is obtained or, (ii) there is no valid prediction made (by Rule 5.5) or, (iii) the global prediction for queries of indices for level \bar{h} has been calculated.*

Rule 5.5 (Invalid Prediction Rule). *A prediction is invalid when it consists of more than one classes which have vote value that is equal or larger than $\max_v - v_e$ where \max_v is the majority vote value and v_e is a discount vote value which is an integer $0 \leq v_e < \max_v - 1$*

In the event the prediction is invalid (C_o of Rule 5.4 is false), the next recall phase is executed with recall requests for the next level are sent to the network, whereas if it is valid (C_o of Rule 5.4 is true) the recall process is terminated with $\hat{r}(h_1)$ as a global prediction $\hat{R}(G)$. As the global prediction is obtained at the first recall, then we achieve the optimal recall prediction (see Property 5.6). For every recall phase, $m_l = |H_l|$ requests for $\{\psi(h_i)\}_{h_i \in H_l}$ are submitted into the network where H_l is a set of nodes at level l and $H_l \in G_R$.

In case the prediction is invalid, the recipient peer z calculates the local prediction $\hat{r}(h)$ using the local function `localRecall`($\psi(h)$) and then replies to the requester x . After receiving the local predictions from m_l peers, the requester performs an aggregation function `agg`($\{r(h_i)\}_{h_i \in H_l}$) to finalise the global prediction $\hat{R}(G)$. Then $\hat{R}(G)$ is validated before terminating (C_o of Rule 5.4 is true) or proceeding (C_o of Rule 5.4 is false) with the lookup for the recall requests at the next level ($l = l + 1$). These processes are stopped when $l > \bar{h}$. The final $\hat{R}(G)$ then is used to obtain a final prediction \mathcal{L} using the function `finalPrediction`(\cdot). This is formally defined in Property 5.6.

Property 5.6 (Optimal Recall Prediction). *An optimal recall is achieved when the valid prediction is obtained during level 1 (the lookup query is found at the root*

node) which only requires a single query message that is sent to the root node and then the root node responds with a valid prediction.

In the case of an optimal prediction with optimal connections^{5.3}, the number of messages is two and recall process require a constant time.

5.4 Experiment

In this section, we empirically evaluate the accuracy and efficiency of our methods in two different problems: image spam detection and email spam detection. All the experiments here were conducted using 10-fold cross-validation and they were run on a PC with Intel Core i-7 2.9GHz and 8Gb memory. The DASMET algorithm is implemented using Java and it is integrated into the Peersim simulator [136]. We initially provide descriptions of problems, followed by the experimental settings which includes the description of the datasets, the DASMET structure for every dataset, and the evaluating classifiers. We also describe the network simulations and the data distributions that were used in the experiments involving distributed algorithms. The performance metrics that were used in evaluating the distributed and centralised algorithms in this experiment are also explained, and the reports from the findings of the experiment discussed. Additionally, at the end, we show the findings on the effect of parameter tuning on the DASMET accuracy and efficiency.

5.4.1 Distributed Image Spam Problem

The image spam detection problem can be defined as follows. Given an image x , the spam detector must classify if the image is spam or ham (legitimate) based on a number of features^{5.4}. Currently, the well-known method to generate these features is the content-based feature extraction. Most of the researches on image spam detection generate these content features using Optical Character Recognition (OCR) techniques, since image spam commonly includes embedded texts. However, the

^{5.3}The lookup process only involves a single hop message.

^{5.4}The term feature is used alternatively with the term attribute in this thesis.

embedded text is always changed to avoid detection. Therefore, in this experiment, we used low level features instead of OCR. Low level features includes corner and edge detection, shape, colour and textures. Al-Duwairi et al. [4] suggests that the textural features are better suited to this problem since the quality of images which are generated by machine are usually not as good as the images produced by human. Image texture is a perceived textural characteristics of an image [190, 147]. Here, we selected three low level descriptors—Haralick descriptor [80], color and edge directionality descriptor (CEDD) [37], and fuzzy color and texture histogram (FCTH) [36], due to their low computation overheads.

5.4.2 Distributed Email Spam Problem

A few examples of email filtering techniques include greylisting, spamtraps, Domain Name System (DNS)-based blackhole lists, and content-based email filtering. In greylisting, a mail transfer agent temporarily rejects any email from unknown sender and accepts the email when it is found legitimate. Spamtraps are email addresses that are used to trap email spam. Incoming email for other email addresses which have the same content with identified spam by spamtraps is blocked, and the IP address of the source is also stored to blacklist the sender.

In the P2P setting, filtering can be done collaboratively where a group of users share their knowledge on email spam. Once email spam is found, the fingerprint of the email spam is generated and shared with all users within the group. The list of email spam's signatures is used to check the incoming email to determine if it is spam. The success of this method, however, depends on the amount of reports from users and the accuracy of the reports. Despite this, the most popular technique these days is content filtering where intelligent content recognition algorithms are used to predict the legitimacy of an email [144, 19, 166]. This technique, however, is computationally expensive and is mostly built for a centralised system. Therefore, here we show the use of DASMET for efficient distributed email spam recognition.

5.4.3 Datasets

The datasets in both image spam and email spam experiments here are publicly available datasets. The dataset that was used for the image spam experiment was obtained from Dredze et al. [60] and Geusebroek et al. [68]. The dataset that was used for the email spam experiment, the Spambase dataset [47], is available from the UCI database [13](which is a public machine learning repository). We used the Weka package [75] to partition the training and test dataset for the 10-fold cross-validation in all experiments.

Dataset for the Image Spam Experiment

We used the personal image dataset [60] and the Amsterdam library of object images (LOI) [68] in this study. This personal image dataset consists of 3,293 images of spam and 1,993 images of ham. Due to difficulties during feature extraction (the feature extraction tool that we used in this experiment were unable to extract the required features), 63 images of spam and 121 images of ham have been removed. To increase the number of ham images, we added 2,000 images of ham from the Amsterdam LOI corpus to increase the size of dataset for our experiments, results in a dataset of 3,872 ham images and 3,266 spam images. The large number of ham images is also useful to investigate the false positive predictions generated by our algorithm. This is important since the high number of false positives is a common problem of the spam filtering—a high number of ham objects is misclassified as spam, even though they contain legitimate content. Figure 5.6 and Figure 5.7 show a few examples of images that were used in this experiment. Table 5.1 provides a summary of the datasets for reference.

Table 5.1: The datasets used in this experiment

Dataset	Ham	Spam
Personal images	1,872	3,266
Amsterdam LOI	2,000	0

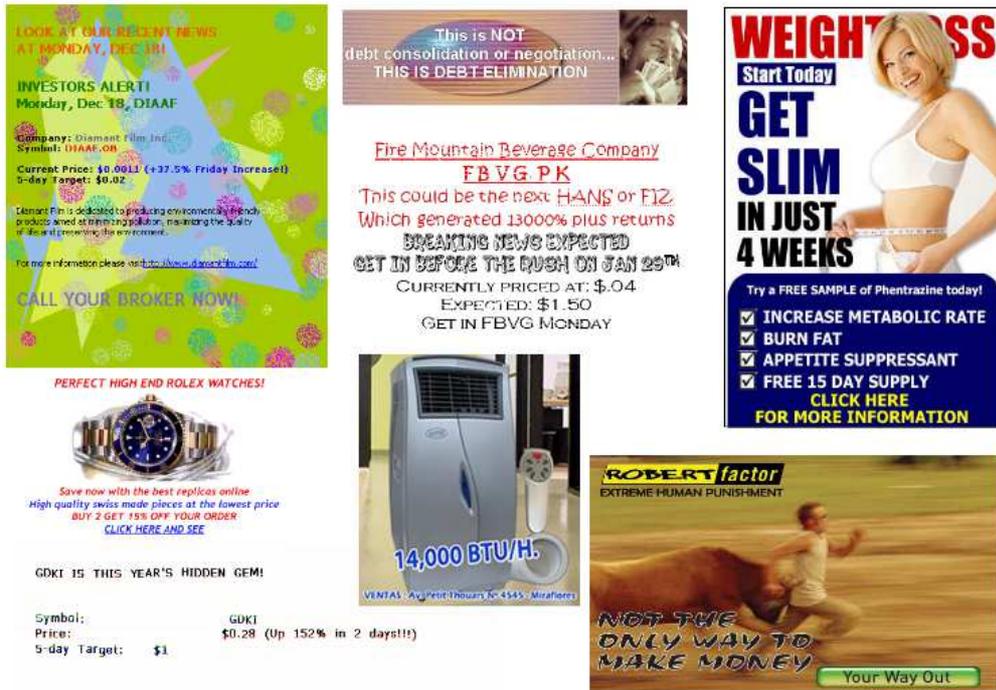


Figure 5.6: Some examples of image spam that are used in this experiment.

Feature Extraction and Pre-processing A tool utilising the Java Advanced Image (JAI) package that is called JFeatureLib [89] was used in this experiment to extract the features. The JFeatureLib produces a feature vector for each image and the resulting feature vectors for all images were transformed into the ARFF format using Weka [75]. Using Weka, we also performed discretisation to transform the continuous values in the dataset into discrete values since our algorithm works only on discrete dataset. The extracted features in this experiment are explained briefly in the following.

Haralick texture descriptor The Haralick descriptor is a second-order statistical feature of texture analysis based on the co-occurrence matrix, which has been effectively used for texture discrimination in biomedical images. The idea is to differentiate the image areas based on texture characteristics. The Haralick descriptor produces fourteen texture features: angular second moment, contrast, correlation,



Figure 5.7: Some examples of image ham that are used in this experiment.

variance, inverse difference moment, sum average, sum variance, sum entropy, entropy, difference variance, difference entropy, information measures of correlation, information measures of correlation and maximum correlation coefficient [80].

CEDD descriptor The CEDD descriptor incorporates color and texture information in histogram form and it produces 144 features, where each feature represents a bin [37]. Initially, the CEDD histogram is divided into six regions that are determined by texture. Then, from each of these six regions, 24 regions are calculated using colour characteristics. Hence, this results in a total of 144 features (6×24).

FCTH descriptor The extraction of the FCTH is similar to the CEDD where a single histogram combines the color and texture information. A fuzzy system was used in this extraction to determine the eight regions based on texture information. This is followed by a similar process in the CEDD, where another 24 regions are

determined based on color information. As a result, the FCTH histogram consists of a sequence of 192 bins which represent a feature vector [36].

Therefore, three separate experiments were conducted using these three different feature sets:

- **Experiment FCTH:** Using the feature set that is extracted using FCTH descriptors.
- **Experiment CEDD:** Using the feature set that is extracted using CEDD descriptors.
- **Experiment Haralick:** Using the feature set that is extracted using Haralick descriptors.

Datasets for the Email Spam Experiment

The Spambase corpus consists of 57 features and 9,203 instances. From these instances, 5,576 instances are ham samples and 3,626 instances are spam samples. The attributes derived from the text in the email is as follows:

- (a) each of the first 48 attributes, represents the percentage of words that match the word ϖ from the email. Each ϖ represents an attribute.
- (b) the next 6 attributes represent the percentage of characters that match the character ζ from the email. Each ζ serves as one attribute out of 6 attributes.
- (c) the average length of uninterrupted sequence of capital letters (one attribute).
- (d) the length of longest uninterrupted sequence of capital letters (one attribute).
- (e) the sum of length of uninterrupted sequence of capital letters (one attribute).

5.4.4 The Classifiers

We compare the accuracy of our algorithm for spam detection with a set of five centralised classifiers: k -nearest neighbour (k -NN), nearest neighbour (1-NN), naive

Bayes, backpropagation neural network (BPNN) and Gaussian radial basis function network (RBFN). We used off-the-shelf components from the Weka package to implement all of these centralised classifiers. We further compare the accuracy with three distributed algorithms: Ivote-DPV [125], ensemble k -NN and ensemble naive Bayes. It is also important to highlight that almost all algorithms in this experiment—1-NN, k -NN, naive Bayes, ensemble k -NN, ensemble naive Bayes, DASMET and P2P-GN) are update-able as they perform incremental learning. Hence, they are feasible for online learning which is useful for P2P networks in dealing large datasets and frequent data updates. The experiment set up of these classifiers are described is as follows.

Out of the five centralised classifiers, three of them requires some parameter tuning. These parameter values are selected by either manual tuning or in consideration to the computational limitations of the hardware used in this experiment. Particularly in the case of BPNN, we set the learning iterations to 50, since it was expensive to process a large number of iterations. The number of hidden neurons in BPNN was set to $\frac{d+|C|}{2}$ where d is the number of attributes (features) and $|C|$ is the number of classes. For RBFN, the classifier uses the k -means clustering algorithm to provide the basis functions and it learns a logistic regression. The number of iterations is set to be unlimited until convergence. For the k -NN classifier, we used $k = 3$ in this experiment.

In the distributed classifier simulations (Ivote-DPV, ensemble k -NN and ensemble naive Bayes), we distributed the training dataset using a joint data distribution and we defined a bound of 50 instances per local data. The training instances in each local dataset are randomly sampled by replacement from the training dataset, using a uniform distribution. The size of local data and the number of peers are reasonable, considering the size of the datasets in each problem. In the Ivote-DPV experiments, we used a pasting algorithm—Ivote—as the local classifier and J48 as the base classifier (as used in Luo et al. [125]). In the Ivote, a small number of instances (called bites) are sampled from the local dataset during each learning iteration. We

set the size of bites to 20% of the dataset size if the dataset size is larger or equal to 100. Otherwise, the size of bites is equal to size of local dataset size. This is to avoid very small bites size, which may result in a high number of iterations before convergence. Here, we used the same setting as in Luo et al. [125], where the parameter $\lambda = 0.002$ was given for the Ivote classifier. For the experiments of ensemble classifiers (ensemble k -NN and ensemble naive Bayes), the local results from all peers were collected at a single peer and these were aggregated using majority voting. All local classifiers (k -NN and naive Bayes) in the ensemble algorithms and J48 algorithm in the Ivote-DPV were implemented using the Weka package.

5.4.5 The P2P-GN and DASMET Structure

The P2P-GN and the DASMET structures are problem-specific. Since there are three different datasets created with three different descriptors in the image spam detection experiment (the Haralick descriptor, the CEDD descriptor and the FCTH descriptor) and one dataset in the email spam detection experiment, we have a total of four different DASMET trees in this experiment.

In order to simplify this experiment, the structures of DASMET were determined based on Rule 5.1, which provides a simple guideline of parameter selection. Nonetheless, at the end of this section, we report the outcome of parameter tuning effects on DASMET's effectiveness and efficiency (see Section 5.4.11). The DASMET structures that we used in this experiment are described in Table 5.2. φ_s values were obtained by applying Rule 5.1 to each problem. For the Haralick experiment, we decided to use $\varphi_s = 3$ (that is the smallest value allowed by Rule 5.1) since the dataset has a very small number of features ($d = 14$). In contrast, we used the highest values allowed by Rule 5.1 ($\varphi_s = \lfloor \log_2 d \rfloor$) in the FCTH, CEDD and email spam experiments, in that d in both of these experiments is high. Note that the values were chosen using tuning-by-hand and based on the idea that a high φ_s may reduce accuracy, while a small φ_s increases communication overheads.

The parameters to create the sub-patterns in the P2P-GN are given as follows. The sub-pattern size, d_s is set to be equal to φ_s and the overlap rate is OV . The values of φ_s and OV for the four experiments in this chapter are given in Table 5.2. Hence, the same number of sub-patterns or number of leaf nodes are generated in both, the P2P-GN and the DASMET in these experiments.

Table 5.2: The P2P-GN and DASMET structures for this experiment.

Dataset	Data dimension (d)	Segment size/ max. children (φ_s)	Overlap rate (OV)	Number of leaf nodes (n_H)	Total nodes (n_R)	Height (h)
Haralick	14	3	2	12	22	3
FCTH	192	7	4	63	85	3
CEDD	144	7	4	47	55	2
Email Spam	57	5	3	27	43	3

Cross-validation for parameter selection or parameter optimisation may be useful to get better results. Here, however, we do not optimise the parameters for this particular problem.

5.4.6 Simulator Set-Up for Distributed Algorithms

The experiments were conducted on a simulation of a 2,000 peers Chord network using the Peersim simulator. We chose this value due to the small size of the datasets in both experiments^{5.5}. Every peer in this simulation stores the link to its k successors and a predecessor. Therefore, every peer has $k + 1$ immediate neighbours. The delay per hop was uniformly distributed between 1 and 20 ms in this experiment.

5.4.7 Performance Metrics

All of these experiments involve binary classification: that is, to classify an item of interest into either spam or ham (legitimate object or non-spam) classes. We used five metrics to evaluate accuracy in these experiments: *accuracy*^{5.6}, *false positive rate*

^{5.5}A distribution of a small size dataset within a large network size results in a large number of duplicates (redundant) of samples across the network.

^{5.6}Accuracy refers to the correctly classified instances

(*FP rate*), *precision*, *true positive rate (TP rate)*^{5.7} and the *F-measure* (F_m)^{5.8} [21]. To evaluate the overheads of the distributed algorithms, two metrics which are affected by the networking related factor were used: number of messages per test and the recall time per test. These performance metrics measure the communication efficiency and time efficiency of the distributed algorithms, particularly for the fully-distributed image spam and email spam detection problems.

The above-mentioned accuracy metrics are further explained in details as follows. The accuracy rate is formally defined in Equation 5.3.

$$\text{accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \quad (5.3)$$

where TP is the true positives, TN is the true negatives, FN is the false negatives and FP is the false positives.

- The true positives are the number of ham objects that are correctly recalled as ham.
- The true negatives are the number of spam objects that are correctly recalled as spam.
- The false negatives are the number of spam objects that are correctly recalled as ham.
- The false positives are the number of ham objects that are incorrectly recalled as spam.

The FP rate, precision, TP rate and F_m are calculated using Equations 5.4, 5.5, 5.6 and 5.7 respectively as the following.

$$\text{FP rate} = \frac{FP}{TN + FP} \quad (5.4)$$

^{5.7}True positive rate is equivalent to recall in the content retrieval context. However, we do not use the term recall here since in the context of this thesis, recall refers to classification or prediction.

^{5.8}F-measure is also known as F-score or F_1 score

$$\text{precision} = \frac{TP}{TP + FP} \quad (5.5)$$

$$\text{TP rate} = \frac{TP}{TP + FN} \quad (5.6)$$

$$F_m = \frac{2 \times \text{precision} \times \text{TP Rate}}{\text{precision} + \text{TP rate}} \quad (5.7)$$

The FP rate measures the rate of ham objects which are incorrectly classified as spam. This measurement is important since the problem with the current spam filters is the high false positive rate, which incorrectly filters out the ham objects from the user's view. The precision measures the fraction of the queries that are correctly recalled as spam from all the queries that classified as spam. The precision is always used together with the TP rate metric which gives the fraction of queries that are correctly recalled as spam from all the queries which are actually spam. Finally, F_m is the harmonic mean of precision and TP rate (a balanced mean between precision and TP rate).

5.4.8 Evaluating Accuracy

The findings of the accuracy evaluation for the image spam detection experiment and the email spam detection experiment are discussed as follows.

Accuracy in Image Spam Detection Experiment

The accuracy results for the FCTH experiment, CEDD experiment and Haralick experiment are reported in Tables 5.3, 5.4 and 5.5, respectively. In general, the result shows that 1-NN performs the best among centralised classifiers (k -NN, BPNN, RBFN and naive Bayes) in these experiments. It has a very high accuracy, low FP rate and high F_m rate. It generally has the lowest FP rate compared to other centralised and distributed classifiers except in the FCTH experiment where k -NN has 0.001 less FP rate than 1-NN.

Within the group of distributed classifiers, the DASMET significantly outperforms other methods in terms of accuracy. The P2P-GN is less accurate than the

Table 5.3: Accuracy on the experiment using FCTH descriptor for image spam detection.

Centralised Classifiers					
Classifier	FP rate	Accuracy	Precision	TP rate	F_m
Naive Bayes	0.107	0.871	0.877	0.846	0.861
1-NN	0.015	0.971	0.983	0.955	0.969
k -NN	0.014	0.959	0.984	0.930	0.956
BPNN	0.029	0.968	0.966	0.965	0.966
RBFN	0.105	0.897	0.873	0.898	0.885
Distributed Classifiers					
Classifier	FP rate	Accuracy	Precision	TP rate	F_m
P2P-GN	0.069	0.941	0.915	0.954	0.934
DASMET	0.018	0.972	0.979	0.960	0.970
Ensemble k -NN	0.138	0.857	0.834	0.851	0.842
Ensemble naive baye	0.125	0.857	0.855	0.836	0.845
Ivote-DPV	0.152	0.822	0.825	0.795	0.810

Table 5.4: Accuracy on the experiment using CEDD descriptor for image spam detection.

Centralised Classifiers					
Classifier	FP rate	Accuracy	Precision	TP rate	F_m
Naive Bayes	0.116	0.862	0.883	0.827	0.855
1-NN	0.009	0.972	0.990	0.949	0.969
k -NN	0.013	0.955	0.987	0.920	0.952
BPNN	0.032	0.972	0.968	0.969	0.969
RBFN	0.111	0.896	0.890	0.883	0.886
Distributed Classifiers					
Classifier	FP rate	Accuracy	Precision	TP rate	F_m
P2P-GN	0.045	0.960	0.946	0.967	0.956
DASMET	0.014	0.983	0.984	0.978	0.981
Ensemble k -NN	0.120	0.827	0.873	0.777	0.822
Ensemble naive Bayes	0.115	0.833	0.878	0.783	0.828
Ivote-DPV	0.152	0.820	0.827	0.790	0.808

Table 5.5: Accuracy on the experiment using Haralick descriptor for image spam detection

Centralised Classifiers					
Classifier	FP rate	Accuracy	Precision	TP rate	F_m
Naive Bayes	0.049	0.951	0.942	0.951	0.946
1-NN	0.005	0.985	0.994	0.973	0.983
k -NN	0.010	0.975	0.989	0.959	0.974
BPNN	0.021	0.978	0.975	0.978	0.976
RBFN	0.048	0.948	0.943	0.943	0.943
Distributed Classifiers					
Classifier	FP rate	Accuracy	Precision	TP rate	F_m
P2P-GN	0.015	0.984	0.982	0.982	0.982
DASMET	0.009	0.987	0.989	0.982	0.986
Ensemble k -NN	0.106	0.911	0.867	0.933	0.899
Ensemble naive Bayes	0.087	0.920	0.894	0.927	0.910
Ivote-DPV	0.164	0.846	0.794	0.859	0.825

DASMET but both are better than the ensemble k -NN, ensemble naive Bayes and Ivote-DPV. The Ivote-DPV performs poorly compared to all other classifiers where its false positive rate for all experiments is the highest and its accuracy rate is the lowest.

The DASMET generally has the best accuracy compared to other classifiers in all experiments. It has comparable accuracy to the centralised 1-NN where its F_m in all experiments are higher than 1-NN, although its FP rate is slightly higher than 1-NN. It achieves 0.97 to 0.987 accuracy rate with 0.97 to 0.986 F_m value. The DASMET's FP rate is higher than the centralised 1-NN in very small magnitude—that is at most 0.004 difference. The high F_m value demonstrates the consistency of the predictions which shows that: the high number of tests that are correctly predicted as spam from all the tests which are classified as spam; and the high fraction of tests that are correctly predicted as spam from all the available spam test objects. The low false positive rate—less than 0.02 for all the experiments using different descriptors—show that DASMET is a good filter since it rarely classifies ham objects as spam, such cases of misclassification often hinder the users viewing the legitimate images which is possibly important or useful to them.

Since the centralised classifiers have all datasets at a single node (where the learning and classification are executed at a single site), this gives an advantage for them to produce a more accurate prediction. The comparison, however, aims to show that the proposed distributed algorithm can produce a comparable accuracy to the centralised implementation of the state-of-the-art algorithms. Nonetheless, since DASMET has shown higher F_m than the state-of-the-art centralised algorithms (1-NN, k -NN, naive Bayes, BPNN and RBFN) for the image spam detection problem, we have proven that our fully-distributed algorithm is indeed competitive with these well-known centralised classifiers. The P2P-GN has also performed competitively with the centralised classifiers particularly naive Bayes and RBFN;—its accuracy and FP rate are better than naive Bayes and RBFN in all experiments. In the Haralick experiment, the P2P-GN has smaller FP rate than RBFN, BPNN and naive Bayes; while, it has higher accuracy and FP rate than naive Bayes, k -NN, BPNN and RBFN.

In relation to the performance of descriptors, our findings again strengthened the relevance of textural features for image spam detection as described in Al-Duwairi et al. [4]. We also found that the Haralick descriptor gives better accuracy for image spam detection than the other two descriptors. This shows the efficiency and effectiveness of the Haralick descriptor for this problem, considering that it is computationally cheaper and produces a smaller number of features, compared to the other two methods.

Accuracy in Email Spam Detection

As shown in Table 5.6, we found that the DASMET produced more accurate results compared to the other distributed methods and has comparable accuracy to the centralised state-of-art algorithms in email spam detection. In this experiment, the DASMET has equal F_m rate and accuracy rate to the 1-NN and is better than the other four centralised classifiers. Its accuracy is very high, at 0.989, with an FP rate of 0.015. The high precision, high TP rate and high F_m further illustrate

the accuracy of the DASMET. Similar to the results in the previous image spam detection experiments, the false positive rate of the DASMET, however, is slightly higher than 1-NN (0.005 difference).

Table 5.6: Accuracy on the spam email dataset

Centralised Classifiers					
Classifier	FP rate	Accuracy	Precision	TP rate	F_m
Naive Bayes	0.099	0.906	0.840	0.914	0.875
1-NN	0.010	0.989	0.985	0.988	0.986
k -NN	0.041	0.945	0.938	0.923	0.930
BPNN	0.045	0.954	0.931	0.952	0.941
RBFN	0.071	0.922	0.889	0.912	0.900
Distributed Classifiers					
Classifier	FP rate	Accuracy	Precision	TP rate	F_m
P2P-GN	0.106	0.923	0.819	0.983	0.893
DASMET	0.015	0.989	0.977	0.996	0.986
Ensemble k -NN	0.190	0.850	0.645	0.963	0.772
Ensemble naive Bayes	0.171	0.859	0.693	0.932	0.795
Ivote-DPV	0.201	0.789	0.663	0.769	0.712

P2P-GN has better accuracy, lower FP rate and higher F_m than the other three distributed classifiers (ensemble k -NN, ensemble naive Bayes and Ivote-DPV). However, it performs worse than the centralised 1-NN, k -NN and BPNN with respect to the accuracy and F_m . Its FP rate are the highest in comparison to all of the centralised classifiers in these experiments.

The DASMET particularly improves the performance of the P2P-GN and this is due to the frequent occurrence of the similar to, or duplicates, of patterns to the stored training patterns which could be a highly possible scenario in real world spam problems. This result shows that the DASMET is the best strategy for distributed algorithms within large-scale distributed system for such problems.

5.4.9 Evaluating Communication Overheads

The number of messages exchanged per test in the ensemble k -NN and ensemble naive Bayes are clearly high (linear in network size), with equivalent magnitude

in all experiments. This is because all peers send their local results to a single requester peer in these algorithms. For that reason, we only report the communication overheads for the P2P-GN, DASMET and the Ivote-DPV in this section.

Since the learning and classification process in the P2P-GN and the DASMET involve finding a key within a DHT system, their communication overheads are influenced by the lookup overhead of the underlying P2P system—the Chord network in this experiment. In contrast, the number of messages per test in the Ivote-DPV relies upon the number of neighbours per peer, since a peer communicates with its immediate neighbours during aggregation. In all experiments, we observed that the distribution of the number of messages per test of the P2P-GN has a much lower standard deviation compared to the DASMET and the Ivote-DPV which are spuriously distributed. These outcomes are reported in the following section.

Evaluating Communication Overheads in Image Spam Detection

The number of messages per test in the image spam experiment are reported in three histograms: Figure 5.8 for the experiment using FCTH feature set, Figure 5.9 for the experiment using CEDD feature set and Figure 5.10 for the experiment using Haralick feature set. Each of the experiments involves a total of 7,138 tests from the 10-fold cross-validation.

In all these experiments, the number of messages per test ranges from 13 messages to 2,000 messages in the Ivote-DPV. The high variance is due to two reasons. First, when there is no conflict during classification (every peer agrees to a single decision), the number of messages per test is minimal; it is equal to the number of neighbours ($k + 1$ where k is the size of the successors list). In this case, a requester peer sends its prediction to its neighbours and if the neighbours also predict similarly, then none of them responds to the requester; hence the DPV process stops within one iteration. Second, when there are high conflicts (high disagreement on the decisions) among peers, the number of DPV iterations equals to the network size, in the worst case.

Figure 5.10 depicts that in all tests the DASMET used fewer than 200 messages in the experiments using the Haralick descriptors. In the experiment using the FCTH descriptor (see Figure 5.8), the number of messages per test in the P2P-GN is normally distributed within a range of 100 to 900 messages, while in the experiment using the CEDD (see Figure 5.9), the number of messages per test is normally distributed between 100 and 700.

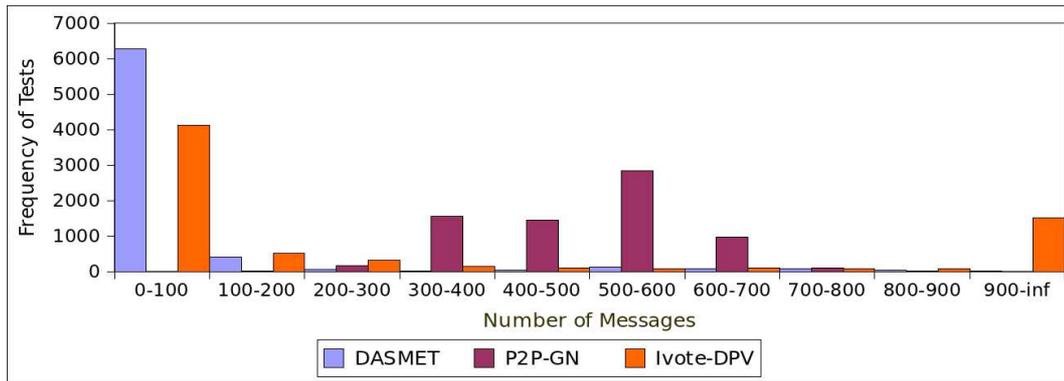


Figure 5.8: Number of messages per test in the FCTH experiment.

The DASMET tests show that the lowest number of messages per test is three in all experiments, while, the highest number of messages per test is 715 in the CEDD experiments, 1,105 in the FCTH experiments and 242 in the Haralick experiments. Although the highest number of messages per test for the DASMET in all experiments is slightly higher than the P2P-GN, the number of messages per test of the DASMET in all experiments are positively skewed. A large proportion of the tests (approximately 87%) consumed less than 100 messages and all of the P2P-GN tests in the CEDD and FCTH experiments required at least 100 messages. Apart from the FCTH experiments, where 27 out of 7,138 tests used more than or equal to 900 messages, none of the DASMET tests in the CEDD and Haralick experiments required 900 or more messages. This shows the efficiency of the DASMET which provides a highly accurate prediction, despite, consuming a low amount of resources.

Due to the small number of features, the communication overhead in the P2P-GN and the DASMET are smaller when using the Haralick descriptors compared to

the CEDD and FCTH descriptors. The experiment using the Ivote-DPV, however, shows differently, as the number of messages per test are found to be greater, with nearly 40% of the tests used at least 900 messages. This suggests that the low number of features may be attributed to the increase in conflicts in the Ivote-DPV classification process, which results in an increase in communication overheads.

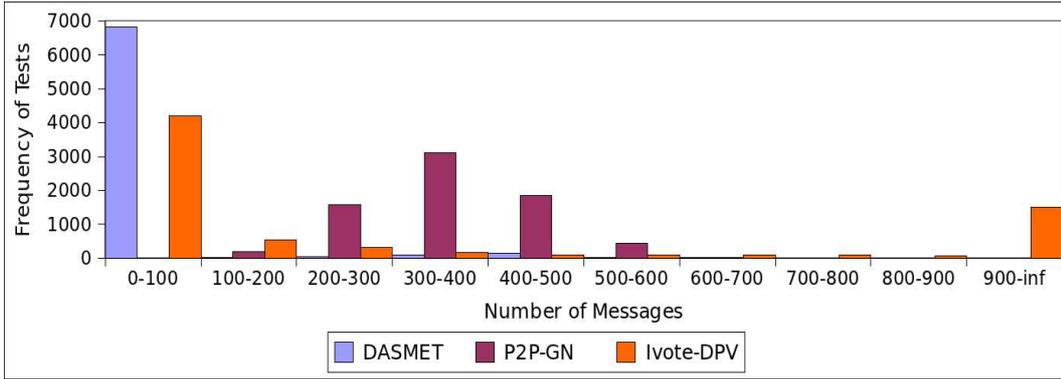


Figure 5.9: Number of messages per test in the CEDD experiment.

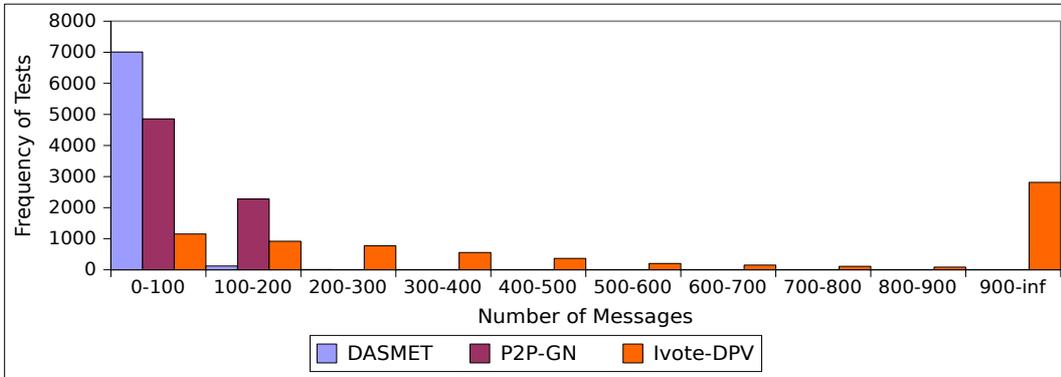


Figure 5.10: Number of messages per test in the Haralick experiment.

Evaluating Communication Overheads in Email Spam Detection

In the email spam experiment, the number of messages per test is reported based on 9,202 observations from the 10-fold cross-validation (see Figure 5.11). A large majority of the tests (95.6% of the tests) required less than 100 messages in the DASMET and we found that none of the tests consumed at least 900 messages.

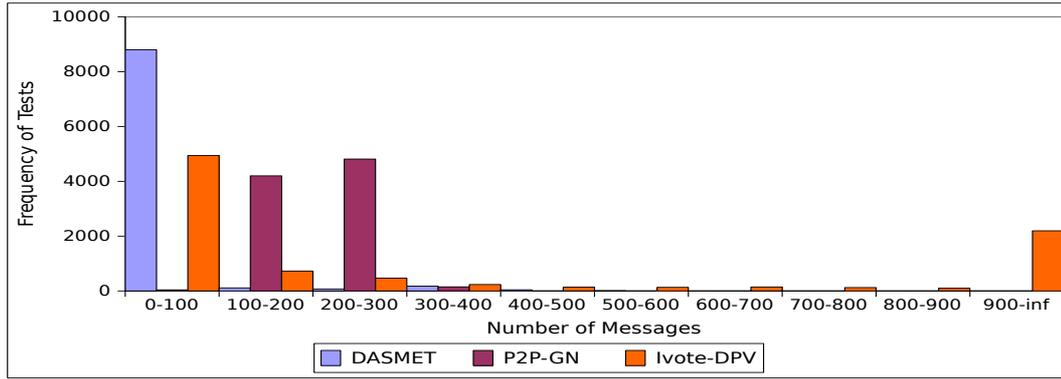


Figure 5.11: Number of messages per test with number of messages in the email spam detection experiment.

In contrast, in the Ivote-DPV, about 53.7% of the tests required less than 100 messages and about 23.8% of the tests required a high magnitude of messages (i.e., 900 messages). The number of messages per test in the P2P-GN, however, are normally distributed between 54 to 351 messages. Only a small number of tests (40 tests) required less than 100 messages, while, 4,200 tests used between 100 to 200 messages and 4,812 tests used between 200 to 300 messages.

The communication overheads in most of the DASMET tests were very low, as they were mostly terminated at the first phase. Hence, they required one lookup message and one response. Every test in the P2P-GN, however, required 27 lookup messages and 27 response messages. Note that the additional message overheads in the DASMET tests and P2P-GN tests resulted from the lookup overheads which in turn depends on the underlying P2P network.

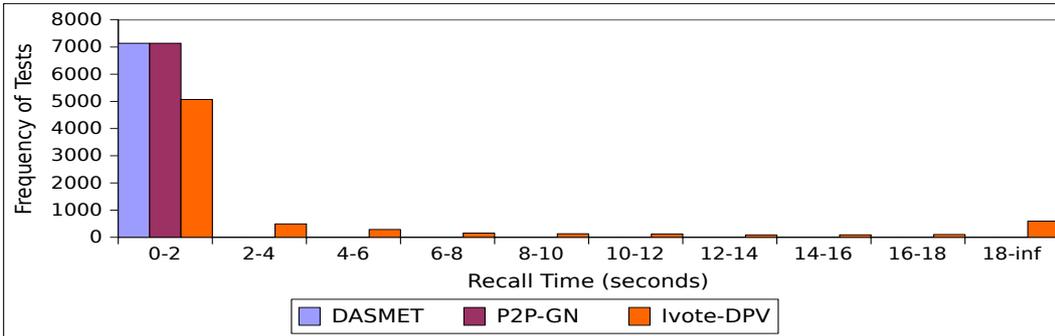
5.4.10 Evaluating Recall Time

In this section, we report the recall time per test in the image spam detection (based on 7,138 observations) and email spam detection (based on 9,202 observations). These numbers of observations are obtained from all the 10-fold cross-validation experiments. Due to the sparseness of recall time distribution, particularly in the Ivote-DPV and the P2P-GN tests, the recall time per test is reported in two different

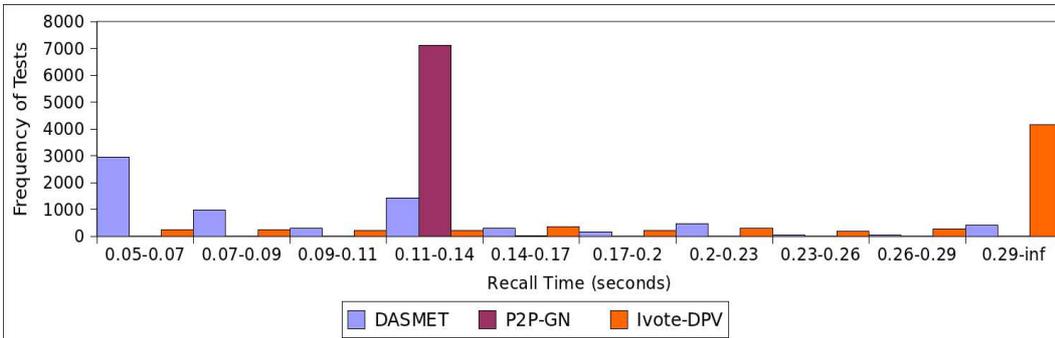
histograms: *Histogram 1* with the high range of 0 – 18 seconds (as shown in Figures 5.12a, 5.13a, 5.14a and 5.15a) and *Histogram 2*, with the short range of 0 – 0.29 seconds (as shown in Figures 5.12b, 5.13b, 5.14b and 5.15b). The Histogram 2 is used to further analyse the comparison of recall time per test between the DASMET and the P2P-GN, since all tests in these two algorithms were completed within less than 1 second. In the DASMET tests, we found that fewer than 1% of tests in the image spam detection experiment and fewer than 2% of tests in the email spam detection experiment took 0.29 seconds or more recall time. However, all of the P2P-GN tests were completed within less than 0.29 seconds.

Evaluating Recall Time in the Image Spam Detection

In Histogram 1 as presented in Figures 5.12a, 5.13a and 5.14a, the results show a similar trend in the FCTH, CEDD and Haralick experiments, respectively.



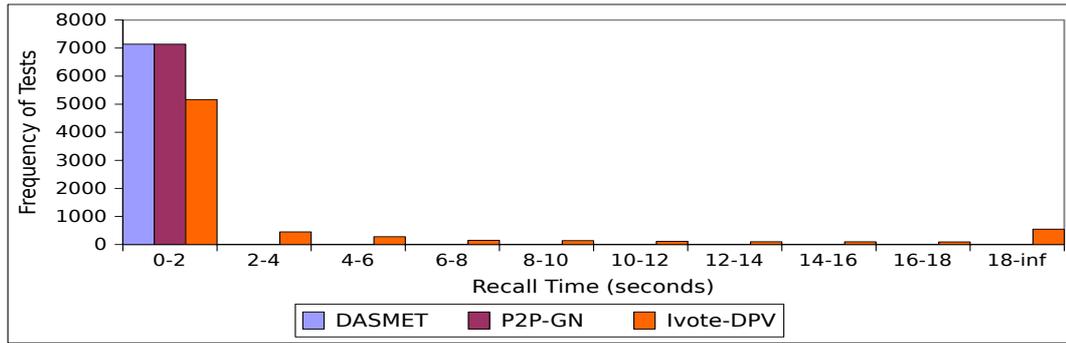
(a) Histogram 1 for recall time per test on the FCTH feature set



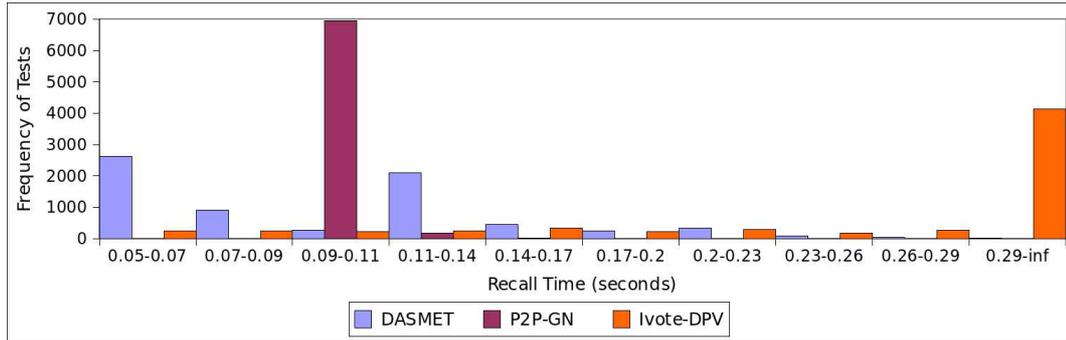
(b) Histogram 2 for recall time test on the FCTH feature set

Figure 5.12: Recall time per test in the FCTH experiment.

We found that all the DASMET tests and all the P2P-GN tests; and about 70%–72% of the Ivote-DPV tests used less than 2 seconds of recall time. Nonetheless,



(a) Histogram 1 for recall time per test on the CEDD feature set

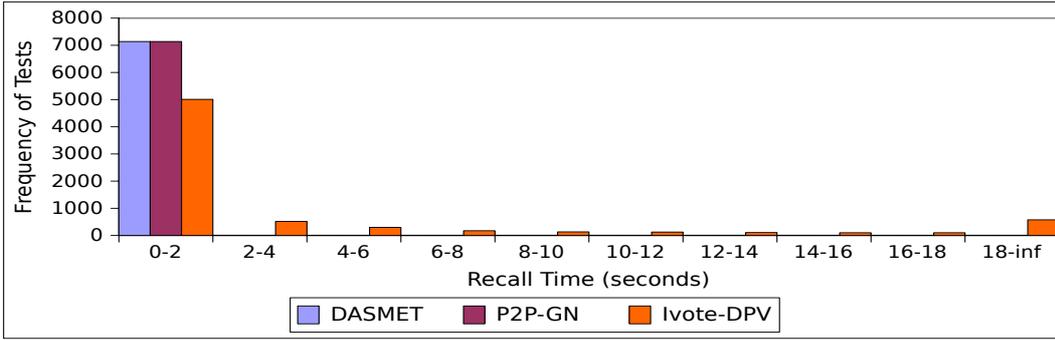


(b) Histogram 2 for recall time per test on the CEDD feature set

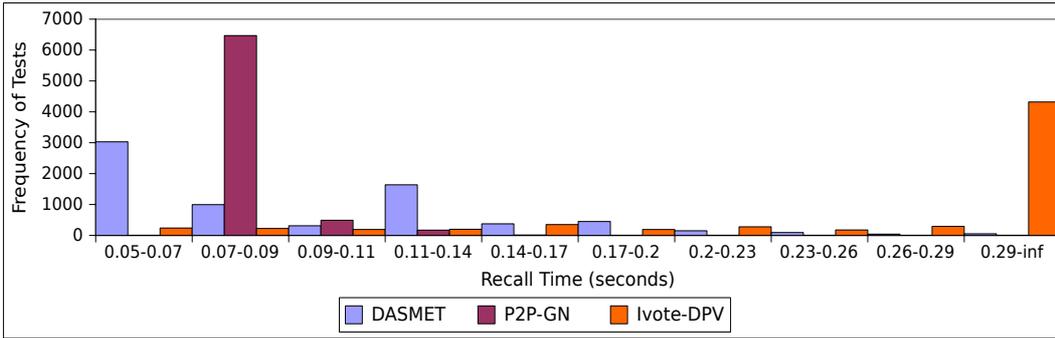
Figure 5.13: Recall time per test in the CEDD experiment.

around 7% to 8% of tests from the Ivote-DPV experiment consumed at least 18 seconds of recall time. Additionally, the highest recall time per test from these experiments was found in the Haralick experiment applying Ivote-DPV, with 47.36 seconds. The reports from Histogram 2 (see Figures 5.12b, 5.13b and 5.14b) depict that about 5.8% of DASMET tests in the FCTH experiment and less than 1% of the DASMET tests in the CEDD and Haralick experiments required at least 0.29 seconds. In the experiments applying the P2P-GN, all of the tests were completed in less than 0.17 seconds. In contrast, about 57% to 60% from the Ivote-DPV tests required at least 0.29 seconds of recall time. This suggests that the DASMET and the P2P-GN are more suitable than the Ivote-DPV for an application which requires a fast response.

Every test in the experiment using either the CEDD or FCTH descriptors and applying P2P-GN completed within 0.09 seconds to less than 0.17 seconds. In the experiment using Haralick descriptor, the P2P-GN required within 0.07 seconds to less than 0.17 seconds to process each test. In other words, 100% of the P2P-GN



(a) Histogram 1 for recall time per test on the Haralick feature set.



(b) Histogram 2 for recall time per test on the Haralick feature set.

Figure 5.14: Recall time per test in the Haralick experiment.

tests were completed in less than 0.17 seconds. However, in the CEDD, Haralick and FCTH experiments, the percentage of the DASMET tests that consumed less than 0.17 seconds of processing time are 89.5%, 88.9% and 83.4%, respectively.

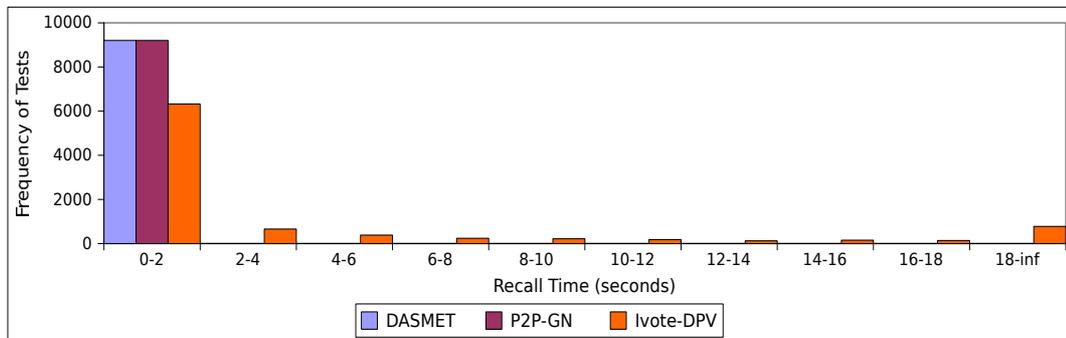
In the other hand, in a group of tests which required less than 0.09 seconds recall time, about 55%, 45% and 56.4% of the DASMET tests required less than 0.09 seconds recall time in the FCTH, CEDD and Haralick experiments, respectively. The group also comprises around 6.9% of the Ivote-DPV tests in the FCTH experiment, 7% of the Ivote tests in the CEDD experiment and 6.4% of the Ivote-DPV tests in the Haralick experiment. In the Haralick experiment, 90.6% of the P2P-GN tests were completed within 0.09 seconds, however, none of them took less than 0.09 seconds in the FCTH and CEDD experiments.

Although the P2P-GN demonstrated a fast recall time in the Haralick experiment, the high proportion of the DASMET tests that used low recall time (< 0.09 seconds) in all of the three experiments, plus the high accuracy of results, demonstrates the ability of the DASMET to provide an accurate prediction with a quick

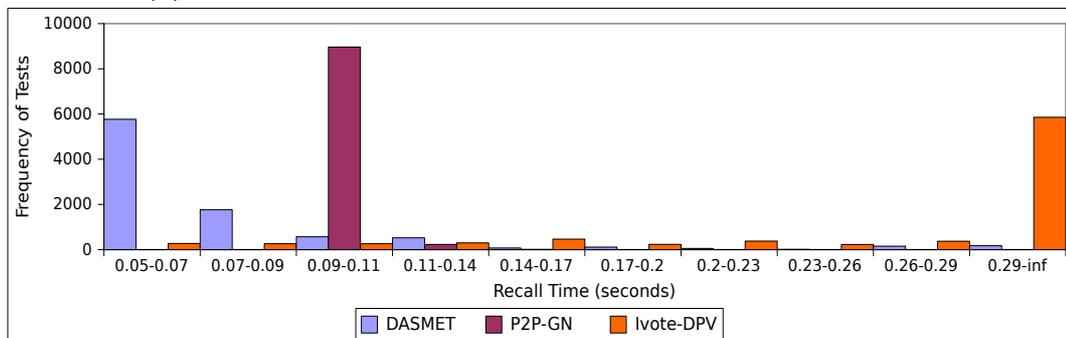
response time. Furthermore, Histogram 2 in Figure 5.14b shows that about 42.4% of the DASMET tests and none of the P2P-GN tests are completed in less than 0.07 seconds in the Haralick experiment. This suggests that the DASMET has improved the recall speed of at least 42.4% tests of the P2P-GN although applying the DASMET may delay about 11% of the tests as a trade-off. This further emphasizes the effectiveness and efficiency of the DASMET.

Evaluating Recall Time in Email Spam Detection

In the email spam detection study, the result in Histogram 1 is generally consistent with the outcomes in the previous image spam detection experiments. We found that in the experiment involving the Ivote-DPV, about 24.1% of the tests required at least 2 seconds of recall time while all tests in the experiment involving the P2P-GN and the DASMET are completed in less than 2 seconds (see Figure 5.15a). This indicates the efficiency of the P2P-GN and DASMET for the email spam detection problem.



(a) Histogram 1 recall time per test on the Spambase dataset



(b) Histogram 2 for recall time per test on the Spambase dataset

Figure 5.15: Recall time per test in the email detection experiment.

Consider Histogram 2 in Figure 5.15b; it depicts that a significantly high percentage (about 63.6%) of the tests in the Ivote-DPV experiment required at least 0.29 seconds recall time and this shows that both the DASMET and the P2P-GN are time-efficient compared to the Ivote-DPV. Every test in the P2P-GN experiment obtained the result in at least 0.09 seconds to under 0.17 seconds. We found that a small percentage of the observations (5.4%) in the experiment applying the DASMET required at least 0.17 seconds. Nonetheless, the result yields that the DASMET has generally improved the recall time of the P2P-GN, as 81.89% of tests required less than 0.09 seconds of recall time (of a total of 9,202 observations), which is considerably large.

The delay of recall time in the DASMET, particularly in the worst case scenario, was caused by \bar{h} factor. The worst case recall time in the DASMET is generally \bar{h} times of the recall time in the P2P-GN. However, the impact of \bar{h} can be significantly reduced when the communication delay per hop is small and vice versa. In the next section, we explore the changes of \bar{h} with varying DASMET's parameters.

5.4.11 Evaluating DASMET Performance with Varying Parameters

In this section, we aim to improve the understanding on the correlation of different parameter values to the effectiveness and efficiency of the DASMET. Here, we explored the effect of different parameters on the DASMET performance in a series of experiments, with regards to three different metrics:

- total number of nodes n_R ,
- height of the tree \bar{h} and
- accuracy, as formally defined in Equation 5.8.

$$\text{accuracy} = \frac{TP + TN}{TP + TN + FP + FN + UD} \quad (5.8)$$

Instead of using Equation 5.3 to represent accuracy in this experiment, we modified Equation 5.8 to include UD which represents the number of tests that return an ‘unknown’ result (when the algorithm is unable to make any prediction). We observed that this situation occurs when φ_s is very large. The metric n_R reflects the storage requirement and the communication overhead of the DASMET—when n_R is high then it requires greater storage overhead and larger communication overhead. The metric \bar{h} , on the other hand, reflects the recall time where a high \bar{h} may result in a large number of recall iterations.

Since the value of d is data-specific and based on the pre-selected feature set, we did not vary this parameter. Hence, we only have one parameter to be tuned, that is φ_s and the resulting OV is $\lceil \frac{\varphi_s}{2} \rceil$. The maximum of φ_s as suggested by Rule 5.1 is $\lceil \log_2 d \rceil$. d values in all experiments are given in Table 5.7.

Table 5.7: Parameter d in all experiments

Experiment	Email Spam	Haralick	FCTH	CEDD
d	57	14	192	144

In this experiment, we increased φ_s so that it is greater than the suggested maximum value. By doing so, we study the performance of the DASMET when φ_s is large. In the Haralick experiment, we slowly increased φ_s value to $d - 1$ starting from $\varphi_s = 2$, therefore φ_s is within the range of $[2, d)$. Since we performed 10-fold cross-validation to obtain each accuracy point, a high number of φ_s values results in a very high number of experiment runs. For instance, in the Haralick experiment with 12 values of φ_s , we conducted $12 * 10 = 120$ simulation runs.

Considering that d in the email spam, FCTH and CEDD experiments are large (d is 57, 192 and 144, respectively), we gradually increased φ_s value from 2.5% to 60% from d by 2.5% instead of using the range $[2, d)$ as used in the Haralick experiment. The resulting 24 values of φ_s in the email spam, FCTH and CEDD experiments are summarised in Table 5.9.

Table 5.8: Parameter φ_s for FCTH, CEDD and email spam experiments

% of d	2.5%	5%	7.5%	10%	12.5%	15%	17.5%	20%
FCTH	5	10	15	20	24	29	34	39
CEDD	4	8	11	15	19	22	26	29
Email Spam	2	3	5	6	8	9	10	128
% of d	22.5%	25%	27.5%	30%	32.5%	35%	37.5%	40%
FCTH	44	48	53	58	63	68	72	77
CEDD	33	37	40	44	48	51	55	58
Email Spam	13	15	16	18	19	20	22	23
% of d	42.5%	45%	47.5%	50%	52.5%	55%	57.5%	60%
FCTH	82	87	92	96	101	106	111	116
CEDD	62	66	69	73	77	80	84	87
Email Spam	25	26	28	29	30	32	33	35

Results

As shown in Figures 5.16 and 5.17, the number of nodes and the height of the DASMET tree decline with an increase in φ_s . This result is generally consistent with Equation 5.2 and Equation 5.2. The minimum value of \bar{h} is 1 while the minimum value of n_R is 3. \bar{h} and n_R converge to their minimum values at a smaller φ_s for the Haralick experiment, compared to the other three experiments, because it has the smallest d . For the same reason, \bar{h} and n_R in the email spam experiment converge to the minimum values at a smaller φ_s than the CEDD experiment, and they subsequently converge at a smaller φ_s in the CEDD experiment than in the FCTH experiment.

The accuracy results (with increasing φ_s values) are presented in Figure 5.18. Generally, the accuracy drops with the growing of φ_s . However, the accuracy metric shows unpredicted results at several points where it dramatically dips to a low value and climbs to a higher value with an increase in φ_s ; this trend appears in all experiments. The Haralick experiment exhibits this significantly where the accuracy drops from 0.955 when $\varphi_s = 9$ to 0.769 when $\varphi_s = 11$ and it suddenly shoots from 0.786 when $\varphi_s = 12$ to 0.936 when $\varphi_s = 13$ (see Figure 5.18a). We observed that in the Haralick experiment and in the CEDD experiment, the accuracy becomes more sensitive to the changes of φ_s . In contrast, the growing φ_s gives less substantial changes to the accuracy in the email spam and the FCTH experiment.

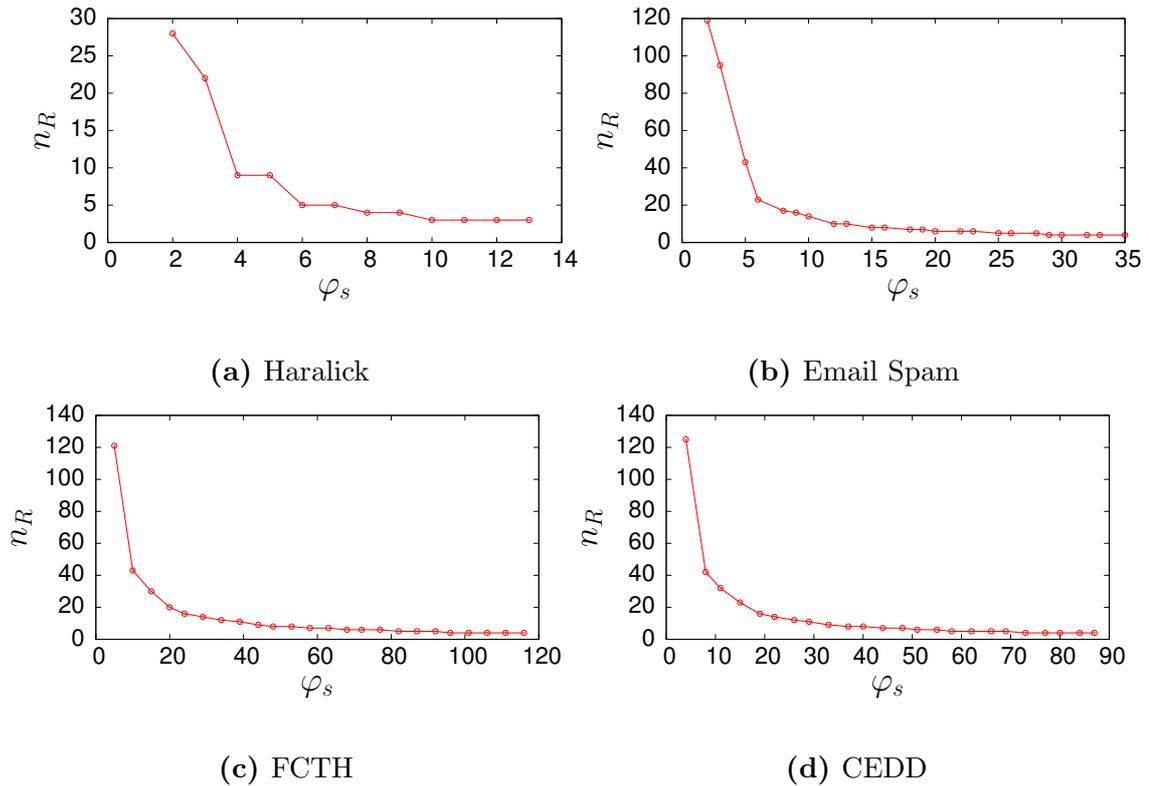


Figure 5.16: n_R with increasing φ_s in the Haralick, email spam, FCTH and CEDD experiment.

This implies that the recall time and communication overheads are improving with increasing φ_s . On the other hand, the accuracy generally declines with increasing φ_s , relative to d value. Generally, a unit increase in φ_s may decrease the accuracy significantly when d is small, while, it may have a less effect on the accuracy when d is large. This effect, however, is variable depending on the application. In the email spam experiment, for example, the accuracy is still high at 0.984 although $\varphi_s = 23$ (40% of d)—a net loss of 0.003 from 0.987 when $\varphi_s = 2$ (2.5% of d). In the previous experiment, we used $\varphi_s = 5$, with resulting accuracy of 0.989 with $n_R = 43$ and $\hbar = 3$ (see Table 5.6 and Table 5.2). Less communication overheads and recall time with the same accuracy can be achieved even if we used $\varphi_s = 10$ where the resulting $n_R = 14$ and $\hbar = 2$. If we define a minimal accuracy rate of 0.98; we can achieve the accuracy at $\varphi_s = 22$ with $n_R = 6$ and $\hbar = 1$.

In the Haralick experiment, we found that the selected value ($\varphi_s = 3$) in the previous experiments (as reported in Table 5.2 and Table 5.5) gave the highest

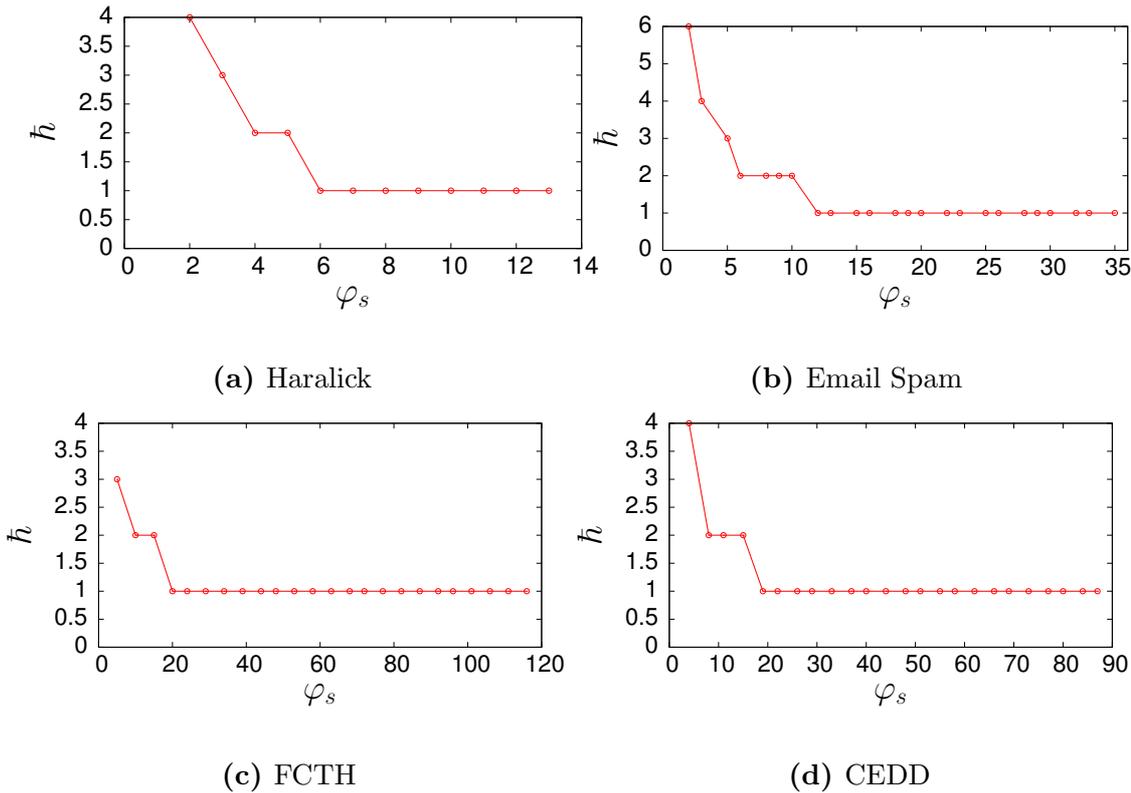


Figure 5.17: \hat{h} with increasing φ_s in the Haralick, email spam, FCTH and CEDD experiment.

accuracy. However, if the minimal accuracy rate is defined as 0.98, this can be achieved at $\varphi_s = 5$. This parameter setting decreases \hat{h} from 3 to 2 and n_R from 22 to 9 nodes. Table 5.3 reports that the DASMET accuracy is 0.972, at $\varphi_s = 7$ in the FCTH experiment. The experimental results show that a minimal accuracy rate of 0.97 can be achieved at $\varphi_s = 20$ where \hat{h} can be reduced to 1 from 2 and n_R can be reduced to 20 from 85.

We found that the selected value in the previous CEDD experiment provides the highest accuracy of 0.983 at $\varphi_s = 7$ (as shown in Table 5.4). The DASMET structure in this experiment has $n_R = 55$ and $\hat{h} = 2$. If the minimal accuracy rate is defined as 0.97 in this experiment, the DASMET can still achieve the desired accuracy even at $\varphi_s = 26$ where $\hat{h} = 1$ and $n_R = 12$. This accuracy is still comparable to the centralised 1-NN and BPNN, and better than other methods, as depicted in Table 5.4.

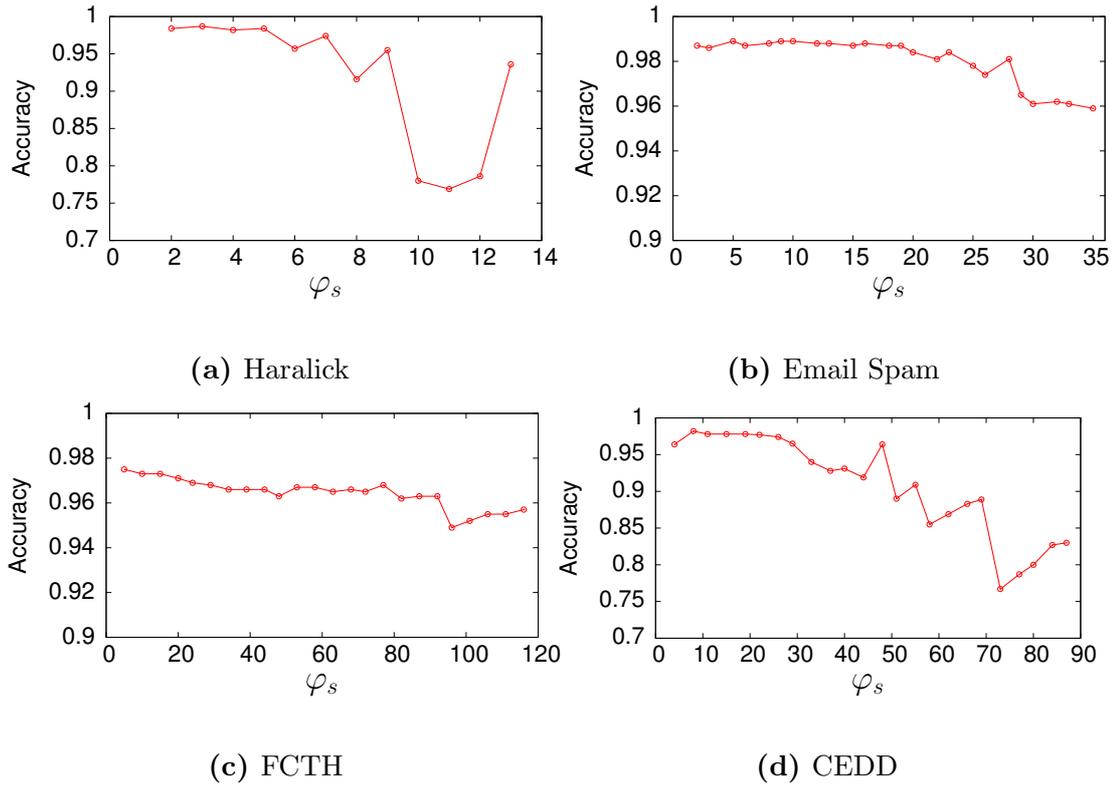


Figure 5.18: Accuracy with increasing φ_s in the Haralick, email spam, FCTH and CEDD experiment.

In summary, it can be deduced that the DASMET performance—as reported in the previous Sections 5.4.8, 5.4.9 and 5.4.10—can be improved by carefully selecting the best parameter. Meanwhile, the experiments demonstrate that the DASMET can still give a high accuracy rate and reasonably low communication overhead and recall time by only using Rule 5.1 as the rule-of-thumb.

Since accuracy is obviously the main priority in the DASMET, the minimal acceptable accuracy rate should be defined for the parameter selection. Nonetheless, the decision must also consider the available resources. For instance, in case the network connection is considerably fast, then \bar{h} metric may not become a factor in the decision. However, if the network connection is really slow, then this metric must be counted, since a single point increment of \bar{h} may considerably delay the whole computation.

5.5 Complexity Analysis

In this section, we analyse the overall complexity at the network level which considers the use of resources in the whole learning or recall operation (involving computational overhead at all participating peers)—starting from the submission of queries at a requester to the end of the learning or recall process. Let d be the data dimension (number of attributes), n_H be the number of sub-patterns or leaf nodes, H_l be a set of DASMET nodes at level- l and \bar{h} be the height of the DASMET tree. We initially discuss the complexity of \bar{h} , which appears as a factor in determining the DASMET's overhead and run-time. This is followed by a discussion on the message and time complexity. The list of notations that are used in this section is provided in Table A.3 of Appendix A.

\bar{h} Complexity

Although DASMET mainly operates in parallel, it also involves a sequential part in its operation. The magnitude of the sequential steps is defined by the height of the tree, that is \bar{h} . Hence, \bar{h} becomes a factor that influences the run-time and communication overhead of the DASMET. Nonetheless, \bar{h} is a small positive integer and \bar{h} is $\mathcal{O}(\log n_H)$ and n_H is $\mathcal{O}(d)$. Since n_H is $\mathcal{O}(d)$, then $\bar{h} = \mathcal{O}(\log d)$. This shows that the height of the tree increases slowly with an increase in data dimension.

5.5.1 Message Complexity

Let n_R be the total number of nodes within the DASMET tree (see Equation 5.2); $M[DASMET]_{learn}$ be the number of messages incurred per learning process; and $M[DASMET]_{recall}$ be the number of messages incurred per recall process. For the implementation on a network with optimal connections^{5.9}, $M[DASMET]_{learn}$ and $M[DASMET]_{recall}$ are $\Theta(n_R)$ messages, while in the implementation within a Chord P2P network, they are $\mathcal{O}(n_R \cdot \log N)$ messages. At the best-case scenario,

^{5.9}Every peer knows the location of each others, so that direct connections among them can be established.

$M[DASMET]_{recall}$ equals two messages since a valid prediction is obtained at the first phase and the lookup function only requires one hop message.

5.5.2 Overall Time Complexity At Network Level

Here, we analyse the overall time complexity in a learning process and a recall process at a network level. The learning time per instance, $t_O[L_{DASMET}]$ and the overall recall time per instance, $t_O[R_{DASMET}]$. $t_O[L_{DASMET}]$ is given in Equation 5.9.

$$t_O[L_{DASMET}] = t_{LQ} + t_{localLearn} \quad (5.9)$$

where $t_{localLearn}$ is the time for local learning processes at leaf nodes and t_{LQ} is the communication time to send `LEARN_REQUEST` messages.

At the worst-case scenario, the recall procedure involves all $(\bar{h} + 1)$ recall phases where the first phase involves nodes at level-0, the second phase involves nodes at level-1 and the last phase involves nodes at level- \bar{h} . In the l th recall phase, $|H_l|$ `RECALL_REQUEST` messages are sent in parallel to H_l (all `DASMET` nodes at level l). Therefore, the upper bound of $t_O[R_{DASMET}]$ is given in Equation 5.10 as below.

$$t_O[R_{DASMET}] = \sum_{l=0}^{\bar{h}-1} (t_{RQ}[l] + t_{localRecall}[l] + t_{RR}[l] + t_{agg}[l]) + t_{RQ}[\bar{h}] + t_{localRecall}[\bar{h}] + t_{RR}[\bar{h}] + t_{agg}[\bar{h}] + t_{fP} \quad (5.10)$$

where $t_{localRecall}[l]$ is the execution time of the function `localRecall(.)` at nodes at level- l ; $t_{agg}[l]$ is the execution time of the function `agg(.)` at a requester to aggregate predictions from all nodes at level- l ; $t_{RQ}[l]$ is the communication time to send `RECALL_REQUEST` messages for all nodes at level- l ; $t_{RR}[l]$ is the communication time to send `RECALL_RESPONSE` messages from all nodes at level- l ; and t_{fP} is the execution time of the function `finalPrediction(.)` at a requester. Note that the function `finalPrediction(.)` is only executed at the last step of the recall procedure.

Then, the best-case recall performance is given in Equation 5.11 as below.

$$t_O[R_{DASMET}]_{best} = t_{RQ}[0] + t_{localRecall}[0] + t_{RR}[0] + t_{agg}[0] + t_{fP} \quad (5.11)$$

To simplify these calculation, we assume that t_{RR} is equal to 1, and t_{RQ} and t_{LQ} are denoted by n_{hops} since these involve a lookup process. $t_{localRecall}$, $t_{localLearn}$ and t_{fP} are denoted by t_1 . Then, t_{agg} is denoted by $\tilde{s} \times t_1$ for \tilde{s} inputs. By applying these assumptions into Equation 5.9 and Equation 5.10, the estimated $t_O[L_{DASMET}]$ and $t_O[R_{DASMET}]$ are given in Equation 5.12 and Equation 5.13, respectively.

$$\begin{aligned} t_O[L_{DASMET}] &= n_{hops} + t_1 \\ &= \mathcal{O}(n_{hops}) \end{aligned} \quad (5.12)$$

$$\begin{aligned} t_O[R_{DASMET}] &= \tilde{h}[n_{hops} + \tilde{s} \cdot t_1 + t_1 + 1] + [n_{hops} + \tilde{s} \cdot t_1 + 2 \cdot t_1 + 1] \\ &= (\tilde{h} + 1)[n_{hops} + \tilde{s} \cdot t_1 + t_1 + 1] + t_1 \end{aligned} \quad (5.13)$$

By substituting $\tilde{h} = \mathcal{O}(\log d)$ and $\tilde{s} = \mathcal{O}(d)$ in Equation 5.13, $t_O[R_{DASMET}]$ is $\mathcal{O}((d + n_{hops}) \log d)$. The aggregation time for level-0, $t_{agg[0]}$ equals t_1 since $\tilde{s} = 1$. Accordingly, by substituting this into Equation 5.11, the estimated $t_O[R_{DASMET}]_{best}$ is given in Equation 5.14.

$$\begin{aligned} t_O[R_{DASMET}]_{best} &= n_{hops} + 3 \cdot t_1 + 1 \\ &= \mathcal{O}(n_{hops}) \end{aligned} \quad (5.14)$$

The estimated communication overhead and recall time in an implementation with optimal connections can simply be obtained by replacing n_{hops} value to 1 in Equation 5.12 and Equation 5.13, while, n_{hops} equals $\mathcal{O}(\log N)$ in an implementation within a Chord network. This is summarised in Table 5.9.

Table 5.9: Summary of the DASMET complexity.

	Overall Computational Complexity	
	Chord network	Optimal Connections
$M[DASMET]_{learn}$	$\mathcal{O}(n_R \cdot \log N)$	$\Theta(n_R)$
$M[DASMET]_{recall}$	$\mathcal{O}(n_R \cdot \log N)$	$\Theta(n_R)$
$t_O[L_{DASMET}]$	$\mathcal{O}(\log N)$	$\mathcal{O}(1)$
$t_O[R_{DASMET}]$	$\mathcal{O}((d + \log N) \log d)$	$\mathcal{O}(d \log d)$

This analysis shows that the DASMET preserves the efficiency and scalability of the P2P-GN. Nonetheless, the advantage of the DASMET is that its best-case performance incurs a significantly low communication overhead and recall run-time,

particularly for a duplicate of a stored pattern—with optimal connections, the best-case recall performance incurs a constant recall time and two messages. Hence, it encourages a fast recall time in the environment with a large number of duplicate and near-duplicate patterns.

5.6 Summary

In this chapter, we have presented an efficient algorithm of a fully-distributed near-duplicates detection,—the DASMET. The algorithm is able to detect exact, near-duplicates and fuzzy patterns in a small number of iterations (at most h iterations). We have reported the application of the DASMET in two spam detection problems (image spam detection and email spam detection). The outcomes of this chapter is as follows:

- The DASMET generally has the best accuracy compared to centralised, state-of-the-art classifiers (naive Bayes, 1-NN, k -NN, RBFN and BPNN) and other distributed classifiers (P2P-GN, ensemble k -NN, ensemble naive Bayes and Ivote-DPV) in the spam detection experiments. In the email spam experiment, it outperforms other methods, however, it has a similar accuracy to the centralised 1-NN.
- The DASMET is effective for image spam detection, with 97% to 99% accuracy in all the experiments, using different feature sets (Haralick descriptor, CEDD descriptor and FCTH descriptor). It is also highly accurate for the email spam problem, with 99% accuracy and less than 0.01 false positive rate. Its false positive rate is less than 0.02 in all spam detection experiments and this implies that it rarely misclassifies legitimate email or images as spam.
- Our experimental result has shown that the P2P-GN has lower accuracy compared to the DASMET, however, it has higher accuracy compared to other distributed methods (ensemble k -NN, ensemble naive Bayes and Ivote-DPV).

- We found that the Haralick descriptor gives a better prediction than the CEDD and FCTH descriptor for the image spam problem in this study.
- We found that the recall time and the communication overhead during recall in the P2P-GN are uniformly distributed, while, those in the Ivote-DPV and the DASMET are spuriously distributed.
- In the worst-case scenario of recall process, the worst communication overhead (highest number of messages per test) in the Ivote-DPV is higher than the worst communication overhead in the DASMET. The worst-case communication overhead in the DASMET is, however, higher than the P2P-GN.
- Nonetheless, the communication overheads in the DASMET is positively skewed where approximately 87% of the DASMET recall tests used less than 100 messages in the CEDD and FCTH experiments, while, none of the P2P-GN recall tests used than 100 messages in those experiments.
- From 9,202 recall tests in the email spam experiment, the percentage of the tests that required less than 100 messages are 4% of the P2P-GN recall tests, 53.7% of the Ivote-DPV recall tests and 95.6% of the DASMET recall tests. This demonstrates that the DASMET improves the communication efficiency in the P2P-GN.
- Our results in the FCTH and CEDD experiments have shown that none of the P2P-GN recall tests was completed in less than 0.09 seconds. However, 55% of the DASMET recall tests in the FCTH experiment and 45% of the DASMET recall tests in the CEDD experiment were completed in less than 0.09 seconds. In the Haralick experiment, the DASMET improves the recall time of at least 42.4% of the P2P-GN tests, but delay about 11% of the tests. These results show the efficiency of the DASMET to provide faster responses in most of the tests compared to the P2P-GN, but with the trade-off of slower responses in a small percentage of the tests.

- The experimental results in the email spam experiment show that 81.8% of the DASMET tests in the email spam problem incurred less than 0.09 seconds recall time. In contrast, none of the P2P-GN recall tests used less than 0.09 seconds recall time. This proves the ability of the DASMET to improve the recall time of the P2P-GN in most of the recall tests in this experiment.
- The experimental results in the experiments with varying DASMET parameters has shown that the best performance of the DASMET—considering the trade-off of the accuracy and the incurred overhead—is possible with the best selection of parameters. The results also showed that the rule-of-thumb in Rule 5.1 is useful to assist in the selection of the parameters which can give a high accuracy rate with a reasonably low communication overhead.
- The worst-case recall time and communication overheads of the DASMET are influenced by h factor which is equal to the height of the DASMET tree. Our theoretical analysis shows that h increases slowly with d .

Chapter 6

Conclusions and Future Work

This thesis has analysed and thoroughly discussed the challenges of pattern recognition within a P2P environment. The main contributions of this thesis are as follows.

The definition of the requirements of the distributed pattern recognition algorithms within P2P networks: This provides a benchmark on the performance of distributed PR algorithms within the networks. In summary, in order to be effective in the context of P2P networks, pattern recognition algorithms need to exhibit the following characteristics: resource-aware, fully-distributed, network-scalable for large networks, online learning and fast response to ad hoc queries, reliable to dynamic networks, asynchronous and invariant to imbalanced data distributions.

A single-cycle pattern recognition method, namely P2P-GN that provides exact global classification but with less computational requirement than the available methods: Distributed Majority Voting (DMV) [207] and Distributed Plurality Voting (DPV) [125] have been proven as effective and efficient foundations for classification within large-scale P2P networks. Classification algorithms that are built using these aggregation methods are guaranteed to converge to an exact global classification^{6.1} representing the whole network [125, 17]. However,

^{6.1}An exact classification equals to the centralised classification which is obtained when the whole data within a network is located at a single site.

this is achieved at a cost of a high number of iterations—which leads to run-time delay, and high communication overheads, particularly when facing difficult problems. The proposed algorithm produces an exact global classification with low run-time and communication overheads during learning and classification; and the communication overhead per peer is independent of the network size. The evaluation on the performance of our algorithms and other algorithms has been done theoretically and empirically on large datasets from the standard machine learning dataset repository UCI database [13]. We have proven that our algorithm’s accuracy is comparable to a number of state-of-the-art algorithms (k -NN, ID3 decision tree, 1-NN, naive Bayes, BPNN, and RBFN).

A major distinction of our method compared to the other approaches is that it builds a single global classifier within the network, instead of building many local classifiers (one at every site). To our knowledge, this is the only work that efficiently and effectively uses a cluster-based algorithm for classification within large and dynamic networks. We showed that there is no need to form a cluster to build this single global classifier, as this can be achieved by distributing the fine-granularity components across the network by using Distributed Hash Table (DHT) technology—which also provides load balancing and efficient linking to these components. The resulting online learning and asynchronous algorithm is proven to be scalable for large-scale distributed networks and it is able to produce a faster response to a recall request compared to Ivote-DPV [125]^{6.2}.

The algorithm is invariant with imbalanced data distribution within P2P network—which is important in solving a real-world problem: Imbalanced class distributions and very small training samples may severely affect the classification accuracy. This often ignored by most distribution classification methods which usually assumes a uniform distribution and a sufficiently large sample size available at every site. We have shown that our proposed method is able to produce

^{6.2}This is subject to the network size larger than the number of attributes.

correct results despite imbalanced data distribution, compared to other distributed algorithms (Ivote-DPV, ensemble ID3, ensemble k -NN, and ensemble naive Bayes).

An economical solution for a scalable distributed algorithm for large datasets: We have shown that resource-efficiency is an important principle for scalability when designing the algorithm in this thesis. In a comparative analysis, we have shown that the P2P-GN is significantly communication-efficient compared to the HGN, DHGN and BPNN. Then, we have experimentally and theoretically demonstrated that the P2P-GN provides an efficient online learning with low run-time at peer level (local) and network level (overall process). The experiments on two large datasets (that were generated by the Waveform data generator (Version 2) and LED data generator which are available in the UCI database) shows that the fully-distributed approach of the P2P-GN reduce the run-time and storage overheads per host in the centralised, state-of-the-art classifiers (BPNN, RBFN, 1-NN, k -NN and ID3).

We have also presented the convergecast recall, which is efficient for high-dimensional problems. This is achieved by reducing the aggregation workload per peer; by aggregating the local predictions iteratively within the tree towards the root node. From the experiment, we have shown that the convergecast recall is resource-efficient per peer, as the recall time per peer is very low. The results effectively demonstrate that our method is scalable and can provide a cheap solution for classification on large datasets.

An efficient fault management scheme for the reliability of our approach within dynamic networks: We devised a local and reactive replication scheme for the P2P-GN, where every peer is aware of its environment and the replication is only executed based on several predefined rules, in order to restrict number of replication processes. By doing so, zero communication is required when the network is in a stable state. The proposed protocol has been evaluated within a simulation of a highly dynamic network and we have shown that it is effective to sustain the system

performance with low overheads. Our experiment has shown that the recovery process only requires a small data transferred per second (only approximately 1% of the total storage size across the network) and the communication overhead for recovery is independent of the network size. The experimental result shows that the proposed scheme manages to maintain the number of replicas across the network to the replication rate r . This demonstrated the efficiency of our approach as it avoids storing unnecessary and expired replicas which might overwhelmed the network over time.

The distributed associative memory tree (DASMET) algorithm to reduce the classification overheads for the type of problem with a high number of duplicate and near-duplicate data: The DASMET algorithm is introduced to improve the communication overheads and recall time in the P2P-GN by detecting the duplicates or closely resembling patterns at the entry point of the system. Hence, it avoids wasting resources for expensive pattern recognition process for this kind of data. This is accomplished without losing the ability to recognize fuzzy data.

We have experimentally demonstrated that the DASMET improves the communication overhead in the P2P-GN. Approximately 87% of the DASMET recall tests used less than 100 messages in the CEDD and FCTH experiments, while, none of the P2P-GN recall tests used the same number of messages. In general, the DASMET has also improved the recall time of the P2P-GN where a large number of tests require much smaller recall time than the P2P-GN. For instance, in the email spam experiment, we have found that 81.89% of tests from a total of 9,202 observations were completed within 0.09 seconds, while, none of the tests was completed within the 0.09 seconds in the P2P-GN.

Compared to the Ivote-DPV, the DASMET is more communication-efficient and time-efficient. In the email spam experiment, 53.7% of the Ivote-DPV recall tests and 95.6% of the DASMET recall tests incurred less than 100 messages. All of the P2P-GN recall tests and the DASMET recall tests were completed in less than

0.29 seconds, while, only approximately 36.4% of the Ivote-DPV recall tests were completed in less than 0.29 seconds.

The DASMET requires several iterations that is bounded by \bar{h} (the height of the tree). \bar{h} grows very slowly with an increase in data dimension (d). Hence, the number of iterations is significantly small in comparison to NN, boosting, DMV-based and DPV-based algorithms.

A fully-distributed intelligent spam detection system The fact that P2P networks have become a popular means to share media files and implement voice or text chat, has attracted spam into such P2P-based applications. Spamming pollutes P2P networks in many ways and these usually encountered in by having a server to filter the spammers. This thesis provides a fully-distributed pattern recognition system using DASMET to detect spam which is cost-efficient and not prone to a single point of failure unlike the server-based systems. We have evaluated our system against centralised state-of-the-art algorithms (NN, k -NN, naive Bayes, BPNN and RBFN) and distributed P2P-based algorithms (Ivote-DPV, ensemble k -NN, ensemble naive Bayes, and P2P-GN). The experimental results show that our method is highly accurate with a 98% to 99% accuracy rate, and incurs a very small number of messages—in the best-case, it requires only two messages per recall test.

The effectiveness of DASMET algorithms in the evaluation on two spam problems (image spam problem and email spam problem) shows the potential of our solution in dealing with spam problems. It has a better accuracy compared to the P2P-GN in these two spam problems. In the image spam experiments, it generally has a better accuracy than other methods (NN, k -NN, naive Bayes, BPNN and RBFN, Ivote-DPV, ensemble k -NN, ensemble naive Bayes and P2P-GN), while in the email spam experiment, it has similar accuracy to 1-NN.

The improvements on the GN, HGN and DHGN First, we significantly reduced the number of communications in the HGN [140] and the DHGN [138]. Second, we eliminated the need for any fusion centre, and third, we also enable an

asynchronous operation which is practical for P2P systems. We have also improved the Graph Neuron's accuracy. These are achieved by having peers recognising joint memory and incorporating hybrid weighting and majority voting approach.

6.1 Potential Applications

By having a fully-distributed pattern recognition within P2P networks, a number of pattern recognition applications such as content filtering, spam detection, and polluted content detection, which were only implementable on a high-performance server can be fully-distributed within a serverless P2P system. This provides an economical solution to build scalable intelligent services for various applications. Moreover, using a distributed pattern recognition may avoid the risk of a single point of failure and build a more fault tolerant system. To show the usefulness of our work in this thesis, we list several examples from the wide range of potential applications of pattern recognition, which are useful for P2P applications, as follows:

Filtering Service: The filtering service involves blocking access to any inappropriate content from any P2P application [174, 120, 148]. As other Web-based media distribution applications, some contents in P2P file sharing systems may be illegal in some organisations or regions. Thus, a filtering service is useful to block any download or access of these illegal materials, in certain regions. The filtering service will also benefit parents in controlling access of pornography, violence and harmful media files for young kids. It is also useful for a company to filter any material which may distract its employees.

Spam Detection: While filtering services, as discussed previously generally, refers to contentious content filtering, spam filtering [116, 135, 86] specifically deals with spam problems. Spam is an action of repetitively sending messages, which is usually aided by a softbot (software robot). Spamming is commonly used as a tool for business promotional or a form of social engineering attack. Instant Messenger

applications such as Yahoo Messenger and Skype currently suffer from spam attacks, which seem difficult to solve.

Polluted Content Detection: Content pollution is a main problem of P2P-based file sharing and video streaming [115, 113, 40, 184, 73, 56]. The file content may be tampered with white noise or advertisements and the file length may be shortened. Polluters may inject a poisoning attack [40] into the metadata of a file to degrade its availability. In P2P video streaming system, the polluters may alter the file chunks to sabotage the streaming media [196].

Music Genre Classification: With a large amount of music files in P2P applications, genre classification is an appealing service provided to users. Much research have been done in genre classification, such as Sanden and Zhang [167], Seo and Lee [172], Tzanetakis and Cook [193], Costa et al. [45]. Genre classification involves labeling music files into several categories (e.g. pop, rock, jazz, metal, country and blues) with members of a category sharing some similarities in terms of instrumentation, pitch and rhythmic structure [193]. This improves content searching and provides better content placement, where users who have the same interest may be placed closer to each other, thus providing more efficient data retrieval.

Anomaly/Attack Detection Service: One of the main security problems in P2P networks is the lack of accountability in the architecture. Any peer can easily connect to any other peer, and these peers are usually anonymous. Therefore, a malicious user can easily inject malware, spam, or viruses into any peer in the network. P2P has a loosely constrained protocol, thus making it easier for malicious users to execute network attacks such as DoS [59], Sybil attack [141, 57]; and attackers may attempt to misroute or drop the messages. For real-time applications like P2PTV or VoIP service, this will significantly delay the audio/video streaming [171]. In client-server approach, the security issues are handled at a reliable server using expensive computations and large storage.

Content Retrieval Service: Instead of a text-based search engine, content retrieval services allows searching by images, audios, and videos [215, 165, 162, 87]. For example, by providing an image of a person who we want to know more about, the search engine can retrieve some possible information about the person, if it is available over the Internet. Examples of client-server based content retrieval applications are Content Based Image Retrieval (CIRES) [87], TinEye [1], and Shazam (audio and video based search engine) [199].

In Amir et al. [5], we have shown that our method is useful for component tracking in a large engineering domain. The large-scale distributed pattern recognition system provides a scalable and more accurate solution, as greater information leads to better decision making. In addition, it provides a more reliable system, which is less risky to a single point of failure; this is a useful characteristic for critical systems.

6.2 Limitations

In this research, we focus on the design and computational aspects of distributed pattern recognition algorithms. Hence, there are several limitations of this research as follows.

The communication overhead of the P2P-GN is $\mathcal{O}(N)$ if the number of attributes is greater than the network size The communication overhead of the P2P-GN is $\mathcal{O}(N)$ when the data dimension (number of attributes) d is greater than the network size N . Nonetheless, this rarely occurs since the network size of a P2P system are usually large.

The run-time (computing time and communication delay) of our proposed scheme depends upon other network related factors in linking the peers. The communication delay between nodes are dependent upon various networking related issues such as physical characteristics, communication technology,

network topology, computer systems and operating systems. Therefore, a real-world implementation may reduce or increase these delays. Although we have considered this issue in designing our proposed scheme (by performing a single cycle learning, parallel processing, low communication overheads and short aggregation steps), a large communication delay may still dominate the overall processing time.

The assumption that the training dataset are processed and labelled prior learning. In this research, we focus on the learning and classification phases of pattern recognition components. This research does not include the data collection and analysis part which involve the process of gathering the raw data and transforming the raw data into processed dataset format (in feature vectors). To be practical for real-world implementations, a fully-distributed data collection and analysis component should be devised and integrated with the proposed pattern recognition scheme.

The assumption that the proposed scheme operates within a safe and secured environment. Considering that P2P network security itself is a big research problem, we do not yet cover the network security aspect in this research. Currently, we assume that the proposed scheme operates within a safe and secured environment. This limits the scheme ability to work within the network which consists of malicious peers and thus, it is threaten by network attacks (e.g., DDoS attack, Sybil attack and Man-in-the-Middle attacks).

The effectiveness and efficiency of the algorithms depend on the correct choice of parameters. The selection of parameter values contribute to the effectiveness and efficiency of the algorithms. The incorrect selection of parameters may result in a high number of leaf nodes, or may restrict the classifier's ability to recognize from a noisy dataset. The high number of leaf nodes may result in a high DASMET tree constructed, and this increases the communication delay which, in the end, slows the overall processing time. For the P2P-GN, this causes a single

peer to combine local results from many peers. The choice of parameters so far is selected heuristically, or by manual-tuning based on previous experiences. A better way of doing this is by performing cross-validations which can be expensive.

The proposed scheme only accepts discrete values. Our algorithm only accepts discrete values; continuous attributes need to be transformed into discrete form through a discretisation process. For sensitive continuous data, this process may reduce the recognition ability.

6.3 Future Research

The results in this thesis provide a foundation for an economical option of large-scale pattern recognition involving large data and distributed environment (with decentralised data and computing resources). Considering that this research involves two large disciplines in computer science i.e., distributed computing and machine learning, there is a wide room for improvement. Furthermore, P2P computing requires different approaches compared to other distributed computing platforms (e.g., grid computing and multi-core computing) since it is self-organising and prone to security problems. Although this research has addressed most of the basic challenges for pattern recognition within P2P networks, additional works are required before it reaches a maturity that leads to successfully integration the proposed scheme with the real world P2P applications. These future works are as follows.

Improving the communication efficiency for high-dimensional problems to deal datasets with a very large number of attributes which can be greater than the network size.

Since our proposed algorithms has a communication overhead which grows linearly with the number of attributes, a very large number of attributes restricts its ability to deal with high-dimensional problem efficiently. The proposed convergecast recall in Chapter 4 has resolved part of this problem by preserving a low computation and

communication overhead per peer. However, the overall communication overheads (considering network-wide processing) remain high. Hence, this issue needs further investigation in future work.

Improving the efficiency (communication and run-time) of the proposed scheme by implementing it on top of overlay networks with more efficient lookups.

In term of the processing speed, implementing the proposed scheme on a standard Chord overlay network [179] results in $\mathcal{O}(\log N)$ processing time since the algorithm's run-time depends on the overlay lookup latency^{6.3}. By implementing our proposed scheme on an overlay P2P network with lower lookup latency, it could provide faster linking between peers and this leads to faster algorithm's run-time. In future, we aim to explore the use of recent improvements of the routing performance in the DHT-based overlay network (e.g., da Hora et al. [50], Shu and Li [175] and Stojnic et al. [180]) to improve the communication efficiency and time efficiency of our algorithm.

Study the integration of the proposed scheme with a fully-distributed data collection and analysis, feature extraction and selection, and training dataset preparation (labelling process).

This research provides a fully-distributed learning and classification scheme which serves as a foundation for a fully-distributed pattern recognition system. Due to the high magnitude of our research problem, we limit the scope of our research to the learning and classification component which is the core of a pattern recognition system and leave the study on the pre-processing component for future work.

To further incorporate the pattern recognition system with fully-distributed process at every part of the system, we plan to extend it with a fully-distributed pre-processing phase. This includes a fully-distributed data collection and analysis, feature extraction and selection and training dataset preparation. A depth study

^{6.3}The lookup latency of a standard Chord overlay network is $\mathcal{O}(\log N)$

is required to investigate a distributed method which involves collaboration among peers for feature analysis and selection, and training data preparation. This may involve investigating the reputation system and distributed aggregation approach to reach consensus among peers on feature selection and training data labelling.

Study the potential network threats and improve the algorithm's robustness against network attacks

More future work that needs attention is the threat from network attacks (e.g., DDoS attack, Sybil attack and Man-in-the-Middle attacks) which are major problems in P2P systems. This, however, is beyond the scope of this thesis. Hence, it needs to be investigated carefully in future.

We plan to explore and model the potential network threats to our scheme. Then, we will study the strengths and weaknesses of our algorithm in dealing with these threats. We will also investigate the enhancement of the algorithm's robustness against the potential attacks.

Optimising parameters selection

Parameters selection is an one-off task—only required at the start of the algorithm. Considering this, though expensive, cross-validation can be a potential option. However, we suggest an investigation of an intelligent optimisation method for the selection of parameters, given many different constraints such as processing time and limit of simultaneous connections allowed per peer.

Improving efficiency of task allocation and scheduling

Some hosts have greater capability than others in P2P networks and hence, these hosts can provide more resources. Efficient scheduling can improve the processing time of the system while reducing the use of resources. A few studies have been conducted for task scheduling, particularly within P2P networks, and these studies can be adopted to improve the efficiency of our algorithm [123, 186].

Other than scheduling, we also plan to adopt the two-tier architecture strategy to improve the efficiency and stability of our algorithm. This is done by selecting a number of supernodes to do heavy-weight tasks such as managing data locations, while the common node performs lightweight processing such as storing replicas. The use of this two-tier strategy can also improve the access to distributed memory. The selection of supernodes can be achieved effectively using optimisation algorithms such as Ant Colony Optimisation (ACO) or Beam-ACO [20] algorithms.

Extending the algorithm to work on distributed multi-label problems

In future, we intend to design a distributed multi-label classifier using this work. Studies related to multi-label classification involve automatically annotating an object with lists of classes [189]. For instance, a “music-into-emotion” classification problem [104]. This is useful considering that the majority of content that is shared within P2P networks are multimedia files, where each of them has the potential to be classified into several classes.

Extending the algorithm to operate on the cloud computing platform.

Cloud computing has emerged as an efficient solution to store large data since organisations no longer need to maintain expensive hardware infrastructure at their sites. Considering the large amount of stored data, there is a need to discover hidden patterns and trends in the large cloud database (data are stored in servers) that leads to useful knowledge for businesses and organisations. Hence, data mining is necessary to facilitate the organisations with such knowledge so that they can be proactive and knowledge-driven. The extension of this research on the cloud computing infrastructure may benefit a large number of organisations that use cloud computing in their operation. The storage elasticity (dynamic data storage that depends on the number of available processors), distributed and parallel approaches of our algorithm allow an efficient processing of the large data within the cloud storage. Thus, it is an important topic for future investigation.

Extending the algorithm to operate on WSNs

WSNs computing provides solution for various applications such as traffic monitoring, air pollution monitoring, natural disaster prevention, forest fire detection and health care monitoring. Considering the significance of WSNs computing in these applications, we plan to extend our algorithm to operate on WSNs platform. Our proposed method is more efficient than the HGN [140] (which was originally designed for WSNs) since the P2P-GN has less number of nodes and communication overheads. In addition, our method is fully-distributed whereas the HGN requires a base station to be its *S&I*. The advantages of our method, however, come with a greater local computational overhead at devices compared to that in the HGN. Nonetheless, as we have shown in Chapter 4, this local computational overhead can be considered low. Considering the continuous improvement on the computational capacity of sensors, the local computational overhead in the P2P-GN shall no longer become an issue which hinder its implementation on WSNs. Therefore, we plan to investigate the feasibility of our algorithm to operate on WSNs in our future work.

6.4 Summary

This thesis has demonstrated the effectiveness and efficiency of our algorithm for pattern recognition involving large-scale distributed networks and large datasets. Here, P2P networks computing emerge as a promising and economical computing platform for alternative in dealing with big data problems. Hence, the proposed method, called the P2P-GN, brings an economical option for a large-scale classification involving large datasets.

The characteristics of P2P networks that we have considered here are asynchronism, server-less, large, dynamic, frequently-updated and have imbalanced data distributions. The resulting algorithm lacks of synchronisation, operates in fully-distributed, scalable to large networks, and works effectively within dynamic networks. It also performs online learning which leads to its scalability for large datasets

and efficiency in dealing with frequent data updates. It also exhibits invariant performance to imbalanced data distributions.

Our algorithm has advantages compared to other pattern recognition algorithms since its memory can be easily partitioned into pieces of either fine-granularity or coarse-granularity. These pieces of memory then are distributed across the network using the DHT system. As a result, the local storage at peers is automatically load-balanced. These pieces can be retrieved efficiently since their locations are deterministic.

The DASMET is designed as an extension of the P2P-GN for such problems with datasets consists of a high number of exact duplicates or near duplicates patterns. The conducted studies have also demonstrated the effectiveness of this methods in solving two spam problems and these studies have shown the effectiveness and efficiency of the DASMET for these type of datasets compared to the P2P-GN.

Also, we have shown significant improvements to the HGN in our approach. The HGN itself is an scalable algorithm for large datasets, but it is semi-distributed, and requires large communication costs particularly when dealing with high-dimensional data. We eliminate the dependency to the server and reduce the number of nodes by incorporating a hybrid weighting and majority voting approach. As a result, this improves the scalability significantly and the server-in-dependency improves the fault-tolerance of the system. This makes it an ideal distributed pattern recognition algorithm within P2P networks.

Our work in this thesis is expected to not only contribute to P2P systems but also for server-to-server systems such as cloud computing. Rapidly growing data makes it painful for organisations upgrading their infrastructure to keep up with sufficient storage requirements. Hence, server-to-server computing may reduce the costs by preserving the available investment through the in-network integration of the available resources with the new ones. Its flexibility for various sizes of networks, while preserving the load balancing, is encouraging in dealing with the problem of Big Data.

Appendix A: Notations

A.1 Notations for Chapter 3

Table A.1: The notations for Chapter 3: P2P-GN For Distributed PR within P2P Networks

Notation	Descriptions
G_P	a P2P network.
N	network size.
\widehat{F}	a set of features, $\widehat{F} = \{f_1, f_2 \cdots f_d\}$.
F_i	a subset of features in \widehat{F} , $F_i \in \widehat{F}$
\mathcal{SF}	a sequence of features.
sf_i	an i th feature of \mathcal{SF}
\widehat{sf}_i	a segment at i th position of \mathcal{SF} .
n	number of training instances.
d	the number of attributes/data dimension where $d = F $.
$\langle \mathbf{x}_j, c_j \rangle$	a training instance.
$\{\langle \mathbf{x}_j, c_j \rangle\}_{j=1}^n$	a training dataset.
\mathbf{x}	a feature vector (an input pattern). $\mathbf{x} = \{x_i\}_{i=1}^d$
\hat{x}_i	an i -th sub-pattern.
\widehat{X}	a set of sub-patterns $\{\hat{x}_i\}_{i=1}^{n_h}$
\mathcal{C}	a set of classes where $c \in \mathcal{C}$.
c	a class label.
u	domain of input (e.g., $\{0, 1\}$ for binary dataset).
OV	overlap rate.
G_R	a set of participating peers in a PR system.
d_s	segment size at a leaf node.

n_H	number of leaf nodes.
$p.ID$	an identifier of a peer p
$p_{suc}[i]$	an i -th successor of a peer p where $p_{suc}[0]$ is the closest successor.
$p_{pre}[i]$	an i -th predecessor of a peer p where $p_{pre}[0]$ is the closest predecessor.
\mathcal{I}	network identifier space.
\mathcal{R}	bias identifier space.
w_v	a set of children of node v
$\hat{\rho}$	$\hat{\rho} = \forall \rho \in G_R.$
n_{hops}	number of hops.
H_{\hbar}	a set of leaf nodes at level \hbar
ψ	bias element's identifier.
\hat{r}	local prediction.
$\hat{R}(G)$	aggregated predictions.
$agg_v(c)$	aggregated vote for class c .
$agg_w(c)$	aggregated weight for class c .
mav_v	the highest aggregated vote.
mav_w	the highest aggregated weight.
\hat{R}_{max}	aggregated predictions with majority vote.
\mathcal{L}	the global prediction.
$\hat{f}(\psi, c)$	frequency of ψ occurred with label c
$w(c)$	weight for class c .
$v(c)$	vote for class c .
\tilde{m}	average number of messages per query.
\tilde{u}	average communication load.
M_{recall}	total number of messages during recall (includes the lookup costs).
$M_{recall}(local)$	local communication complexity at the peer level during recall procedure.
$M_{learn}(local)$	local communication complexity at the peer level during learning procedure.
$M_{recall}(overall)$	overall communication complexity at the network level during recall procedure.
$M_{learn}(overall)$	overall communication complexity at the network level

	during learning procedure.
n_{recall}	total number of recall tests.
\tilde{l}_{recall}	total communication load during recall.
\tilde{t}_{recall}	average time taken for recall per instance.
M_{LQ}	number of hops to send a <code>LEARN_REQUEST</code> message.
M_{RR}	number of hops to send a <code>RECALL_RESPONSE</code> message.
M_{RQ}	number of hops to send a <code>RECALL_REQUEST</code> message.

A.2 Notations for Chapter 4

Table A.2: The notations for Chapter 4: A P2P Classification for Large Datasets

Notation	Description
N	network size.
G_P	a P2P network.
G_R	a set of participating peers in a PR system.
d_s	segment size at a leaf node.
m	maximum of children per node.
OV	overlap rate.
n_H	number of leaf nodes.
n_R	number of vertices in the convergecast tree.
h	height of the convergecast tree.
d	number of attributes/data dimension.
q	number of stored instances.
\mathcal{C}	a set of classes where $c_i \in \mathcal{C}$.
c	a class label.
u	domain of input (e.g., $\{0, 1\}$ for binary dataset).
n_b	estimated size of the bias array.
\mathfrak{P}	a set of peers that are selected by <code>elect(.)</code> function.
$\hat{\mathcal{S}}$	a set of bias identifiers.
$\hat{\mathcal{S}}'$	a subset of $\hat{\mathcal{S}}$.
$n_{\mathcal{I}}$	number of iterations.
$M[BPNN]_{learn}$	number of messages in a BPNN learning.

$M[BPNN]_{recall}$	number of messages in a BPNN recall.
$M[HGN]_{learn/recall}$	number of messages in a learning or recall of the HGN.
$h[HGN]$	height of the HGN structure.
$M[DHGN]_{learn/recall}$	number of messages in a learning or recall of the DHGN.
$h[DHGN]$	height of a DHGN sub-net.
$M[P2PGN]_{flat,recall}$	number of messages in flat recall.
$M[P2PGN]_{convergecast,recall}$	number of messages in convergecast recall.
$M[P2PGN]_{learning}$	number of messages during learning.
$t[L]_{\mu}$	average learning computation time.
$t[R]_{\mu}$	average recall computation time.
S	storage overhead at a single host in the centralised architecture.
$t[P]_{max}$	maximum processing time taken per host for an instance.
S_{μ}	average storage overhead per host in a distributed system.
$t_{l,\mathbf{x}}$	training time of an instance \mathbf{x} .
\mathcal{X}_{learn}	a set of training instances.
$t_{r,\mathbf{x}}$	the recall time of an instance \mathbf{x} .
\mathcal{X}_{recall}	a set of test instances.
$t[P]_{max}$	the highest per host processing time
$S[p]$	storage overhead at a host p .
$t[p]$	processing time at a host p .
t_L	local learning time per instance.
t_R	local recall time per instance.
\tilde{s}	the number of local classifiers which are inputs in an aggregation process.
n_b	the number of bias elements per leaf node.
$n_{b,total}$	the total number of bias elements within the system.
$n_{b,total}/P$	the average number of bias elements per peer.
n_{OB}	the original number of bias elements (the number of bias elements in a stable network).
$d_{s,min}$	the minimum value of d_s .

$d_{s,max}$	the maximum value of d_s .
$t_O[L]$	overall learning time.
$t_O[R_{flat}]$	overall recall time in the flat recall approach.
$t_O[R_{convergecast}]$	overall recall time in the convergecast recall.
$t_{localLearn}$	execution time of function <code>localLearning(.)</code> .
$t_{localRecall}$	execution time of function <code>localRecall(.)</code> .
t_{LQ}	communication time to send a <code>LEARN_REQUEST</code> message
t_{agg}	execution time of function <code>agg(.)</code> .
t_{fP}	execution time of function <code>finalPrediction(.)</code> .
t_{RQ}	communication time to send a <code>RECALL_REQUEST</code> message.
t_{RQ}	communication time to send a <code>RECALL_REQUEST</code> message.
t_{RR}	communication time to send a <code>RECALL_RESPONSE</code> message.
t_{FR}	communication time to send a <code>FORWARD_RESPONSE</code> message.
t_{FQ}	communication time to send a <code>FORWARD_REQUEST</code> message.
$B_A[t]$	the number of available bias elements at time t divided by n_{OB} .
r	replication rate.
$\Gamma(p)$	neighbourhood of a node p .
$p.ID$	an identifier of a peer p .
$p_{suc}[i]$	an i -th successor of a peer p where $p_{succ}[0]$ is the closest successor.
$p_{pre}[i]$	an i -th predecessor of a peer p where $p_{pre}[0]$ is the closest predecessor.
Θ	master storage.
Ξ	replica storage.
$p.\Theta$	master storage of a peer p .
$p.\Xi$	replica storage of a peer p .
\tilde{R}	a subset of bias elements in Ξ .
\hat{T}	a subset of bias elements in Θ .
$SList$	successor list.
$SList'$	ex-successor list.
$SList_{pre}$	the successor list of the immediate predecessor.
$AList$	inter-update predecessor list.
$PList$	predecessor list.
$PList'$	ex-predecessor list.
$SList_{\widehat{pre}}$	the successor list of a predecessor \widehat{pre} .

$NewSuc$	$SList \not\subseteq SList_{pre}$
$DeadA$	$AList \not\subseteq PList$
$ObsPre$	$PList' \not\subseteq PList$
$MPre$	$PList \cap PList'$
$x.index$	the index of x in the list.
$RPre$	$\{s \in ObsPre; z \in MPre; s, z \in PList' : s.index > z.index\}$.
$JPre$	$ObsPre \not\subseteq RPre$
$ChangedSuc$	$SList \not\subseteq SList'$
\tilde{U}	supernode selection criteria.
\widetilde{U}_{min}	minimum requirement for every criterion in \tilde{U}
\widetilde{W}	a set of criteria weight.
\tilde{u}_i	i th criterion in \tilde{U} .
$\tilde{u}_{min,i}$	minimum capability for \tilde{u}_i .
\tilde{w}_i	weight for i th criterion.
$\tilde{v}_{p,i}$	normalised value of criterion i of a peer p .
$Score_p$	score of a peer p calculated using simple additive method.
\hat{P}	a set of selected supernodes.
$t_{lastupdate}$	the last periodic recovery time.
δ	a predefined time interval to update the replicas.
MB_A	availability rate of master bias elements.
RB_A	availability rate of replica bias elements.
M/t	the number of messages per second in recovery process.
L/t	the data transferred per second.
$M_{routine}/p$	the average number of messages per peer for every routine recovery.
n_{MB}	the available number of master bias elements in the system.
n_{RB}	the number of replicas in the system.
t_{total}	length of simulation time.
n_M	total number of messages.
t_{load}	total size of the transferred data (in MB).
$M_{routine}$	the number of messages in an execution of a <code>routineRecovery(.)</code> .
p'	a bootstrapping peer.

A.3 Notations for Chapter 5

Table A.3: The notations for Chapter 5: Distributed Spam Detection

Notation	Descriptions
D_R	a DASMET tree $D_R = \{V_R, E_R\}$.
V_R	a set of nodes in D_R , $V_R = \{v_i\}_{i=1}^{n_R}$.
E_R	a set of edges in D_R .
G_R	a set of peers which hold roles of V_R .
ψ	a bias element's identifier.
$\psi(h)$	a bias element's identifier that is sent for recall at a node $h \in G_R$.
$\hat{r}(h)$	a local prediction at a node $h \in G_R$.
$\hat{R}(G)$	aggregated predictions.
$\hat{\psi}$	a set of bias identifiers.
$\langle \psi, b \rangle$	a learn query where b is the associated label.
d_s	the sub-pattern size.
φ_s	the maximum number of children of each node, and equals d_s .
d	number of attributes/data dimension.
H_l	a set of nodes at level l where $H_{\bar{h}}$ is a set of leaf nodes and H_0 consists of a root node.
\bar{h}	the height of D_R .
m_l	$ H_l $. The number of nodes at level l .
\mathcal{L}	global prediction.
w_v	a set of children of node v
$\hat{\rho}$	$\hat{\rho} = \forall \rho \in G_R$.
q_i	a query (consists of an identifier) for a bias element i .
n_{hops}	number of hops.
N	network size.
$t_{localLearn}$	execution time of function <code>localLearning(.)</code> .
$t_{localRecall}$	execution time of function <code>localRecall(.)</code> .
t_{LQ}	communication time to send a <code>LEARN_REQUEST</code> message.
t_{agg}	execution time of function <code>agg(.)</code> .
t_{fP}	execution time of function <code>finalPrediction(.)</code> .
t_{RQ}	communication time to send a <code>RECALL_REQUEST</code> message.
t_{RR}	communication time to send a <code>RECALL_RESPONSE</code> message.

n_c	the number of child nodes.
\tilde{s}	the number of local classifiers which are inputs in an aggregation process.
$M[DASMET]_{learn}$	number of messages in per instance learning operation.
$M[DASMET]_{recall}$	number of messages in per instance recall operation.
$t_O[L_{DASMET}]$	overall learning time per instance.
$t_O[R_{DASMET}]$	overall recall time per instance.
$t_O[R_{DASMET}]_{best}$	the best-case of overall recall time per instance.

Appendix B: P2P-GN

B.1 Generating Sub-patterns

Algorithm B.1. createSub-Patterns

```
1: Input:  $\mathbf{x} = \{x_1, x_2, \dots, x_d\}$  (input pattern),  $d = |\mathbf{x}|$  (pattern dimension),  $d_s$ 
   (sub-pattern size),  $OV$  (overlap position)
2:
3:  $j \leftarrow 1$  (sub-pattern's index)
4:  $pos \leftarrow 1$  (point to the first position in  $\mathbf{x}$ ).
5: while  $pos \leq d$  do
6:   Create new sub-pattern  $\hat{x}_j$ .
7:    $i \leftarrow 0$ 
8:   for  $i < d_s$  do
9:     if  $pos \leq d$  then
10:      Put the value  $x_{pos}$  into  $\hat{x}_j$ 
11:     else
12:      Put padding value  $\#$  into  $\hat{x}_j$ 
13:     end if
14:      $pos \leftarrow pos + 1$ 
15:      $i \leftarrow i + 1$ 
16:   end for
17:   Put  $\hat{x}_j$  into  $\widehat{X}$ 
18:    $pos \leftarrow pos - OV$ 
19:    $j \leftarrow j + 1$ 
20: end while
```

Algorithm B.1 creates sub-patterns from \mathbf{x} with parameters d_s and OV . The process starts from the first position in \mathbf{x} which is x_1 where the position is $pos = 1$. We create a new sub-pattern \hat{x}_j with size d_s for j th sub-pattern in the steps from Line 7 to Line 16 of Algorithm B.1. Here, a padding value $\#$ is added into \hat{x}_j if pos has passed the last position in \mathbf{x} ($pos > d$). Otherwise, the value of x_{pos} is added into \hat{x}_j . The current position, pos is incremented for each time a new value is added into

\hat{x}_j as shown in Line 14. After completing the process of creating the sub-pattern, we step back at OV positions from the current position where $pos = pos - OV$. If the current position has passed the last position in \mathbf{x} ($pos > d$), then the process stops here. Otherwise, the process of creating a new sub-pattern begins, and this process is repeated until $pos > d$. Finally, we obtain a set of sub-patterns \widehat{X} .

Lemma B.1. *Let d be the number of attributes (the size of an input pattern), d_s be the sub-pattern size, and OV be the overlap rate. The number of sub-patterns or leaf nodes, n_H , is given by the function $f(d, d_s, OV) = \lceil \frac{d-d_s}{d_s-OV} + 1 \rceil$ for all integers d, d_s , and OV where $d > d_s > OV$, $d_s > 0$, and $OV \geq 0$.*

Proof. Iteratively, $f(d, d_s, OV) = f(d - k(d_s - OV), d_s, OV) + k$ where at the k th iteration, we have found k sub-patterns. However, this only holds for integers $k \leq k^*$ where k^* is the smallest integer such that $0 < d - k^*(d_s - OV) \leq d_s$. Since, we have three cases:

$$\begin{aligned} \text{if } d - k(d_s - OV) \leq 0 & \quad \text{then } f(d - k(d_s - OV), d_s, OV) = 0, \\ \text{if } 0 < d - k(d_s - OV) \leq d_s & \quad \text{then } f(d - k(d_s - OV), d_s, OV) = 1, \text{ for exactly one } k \\ \text{if } d - k(d_s - OV) > d_s & \quad \text{then } f(d - k(d_s - OV), d_s, OV) > 1. \end{aligned}$$

Thus, we find k^* as follows.

$$\begin{aligned} d - k(d_s - OV) &\leq d_s \\ \Rightarrow k &\geq \frac{d-d_s}{d_s-OV} \\ \Rightarrow k^* &= \lceil \frac{d-d_s}{d_s-OV} \rceil \end{aligned}$$

Therefore,

$$\begin{aligned} f(d, d_s, OV) &= f(d - k^*(d_s - OV), d_s, OV) + k^* \\ &= 1 + k^* \\ &= 1 + \lceil \frac{d-d_s}{d_s-OV} \rceil. \end{aligned}$$

The result follows. □

B.2 An example of the P2P-GN recall process

Figure B.1 shows the sequences of processes during recall in the P2P-GN. The requester peer x sends recall requests to all leaf nodes p_1 , p_2 , p_3 and p_4 for bias identifiers ψ_1 , ψ_2 , ψ_3 and ψ_4 in parallel at time $t = 0$. Upon receiving these recall requests, these leaf nodes compute the local results simultaneously using function $\text{localRecall}(\cdot)$ at time $t = 1$ and send their responses to the peer x at time $t = 2$. At time $t = 3$, the peer x then aggregates these results using function $\text{agg}(\cdot)$ into $\hat{R}(G)$ and makes a final prediction \mathcal{L} using the function $\text{finalPrediction}(\cdot)$.

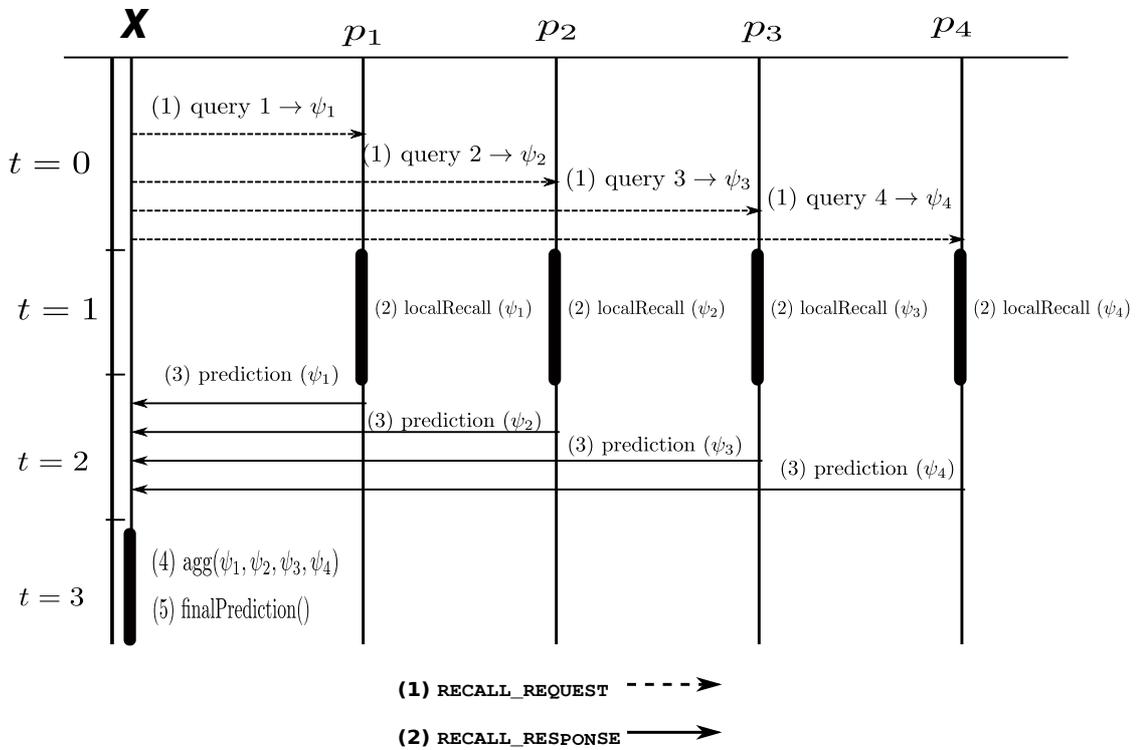


Figure B.1: An example of a P2P-GN recall process. The queries ψ_1 , ψ_2 , ψ_3 and ψ_4 are submitted to leaf nodes p_1 , p_2 , p_3 and p_4 simultaneously. The final decision is made upon the aggregation is completed at peer x .

Appendix C: Convergecast Recall

C.1 Maximum Simultaneous Connections Per Peer

Table C.1 provides examples of good setting for the maximum number of global connections per peer for a given time based on the upload speed. The upload speed is used here, considering in the P2P systems, upload stream is an important measure of a peer contribution to the network. Basically, the peer which provides a high upload speed receives a faster download rate.

Table C.1: The Suggested Maximum Global Connection Per Peer

Upload Speed (kbits/s)	Maximum Global Connection
128 3G(HSDPA)	40
128	80
256	130
512	200
768	250
1024	300
2048	300
5000	360

Source: http://wiki.vuze.com/w/Good_settings

C.2 Tree Construction

The process of constructing a convergecast tree is recursive and it starts from a root node. Let level be 0, $\widehat{X} = \{\hat{x}_i\}_{i=1}^{n_h}$ be a set of sub-patterns, w be n_H , m be the maximum number of children of each node and d_s be the segment size at a leaf

node. All segments in \widehat{X} are initially assigned to the root node V where the following steps in function $\text{constructTree}(\text{level}, w, \widehat{X}, m, H)$ as explained in Algorithm C.1 are executed. Note that \widehat{X} is generated by Algorithm B.1 of Appendix B.

Algorithm C.1. $\text{constructTree}(\text{level}, w, \widehat{X}, m)$

```

1: level ← level + 1
2: if  $w \leq m$  then
3:   create  $w$  leaf nodes  $H = \{h_1, h_2, \dots, h_w\}$ 
4:   assign  $\hat{x}_i \in \widehat{X}$  to leaf node  $h_i \in H$ 
5: else
6:    $n_c \leftarrow 0$ 
7:   if  $\lfloor \frac{w}{m} \rfloor < m$  then
8:      $n_c \leftarrow \lfloor \frac{w}{m} \rfloor + 1$ 
9:   else
10:     $n_c \leftarrow m$ 
11:   end if
12:    $n \leftarrow \lfloor \frac{w}{n_c} \rfloor$ 
13:   balance ←  $w - (n \times n_c)$ 
14:    $y \leftarrow 0$ 
15:   while  $y < n_c$  do
16:      $w \leftarrow n$ 
17:     if balance > 0 then
18:        $w \leftarrow w + 1$ 
19:       balance ← balance - 1
20:     end if
21:     create a child node  $h_i$ 
22:     assign  $w$  segments,  $\widehat{X} = \{\hat{x}_1, \hat{x}_2, \dots, \hat{x}_w\}$  to  $h_i$ 
23:      $\text{constructTree}(\text{level}, w, \widehat{X}, m)$ 
24:      $y \leftarrow y + 1$ 
25:   end while
26: end if

```

The node firstly determines whether it should expand the tree or not. In case that w is less or equal to m , then the node creates w leaf nodes and assigns one segment per leaf node; this completes the process. Otherwise, it determines the number of children n_c using Equation 1 as below.

$$n_c = \begin{cases} m & \text{if } \lfloor \frac{w}{m} \rfloor \geq m \\ \lfloor \frac{w}{m} \rfloor + 1 & \text{otherwise} \end{cases} \quad (1)$$

Next, it creates n_c child nodes and distributes the available segments to these child nodes using greedy approach. Upon receiving w segments from its parent, every

child node then executes Algorithm `constructTree(level,w, \widehat{X} ,m)`. This process is executed recursively until $w \leq m$.

C.3 Discussion on Communication Overheads in the BPNN, HGN, DHGN and P2P-GN

To explain the communication overheads in the BPNN, HGN, DHGN and P2P-GN, a brief discussion on the communications involve in these algorithms are given as follows.

BPNN The BPNN in this analysis has only one layer of hidden neurons in which the number of hidden neurons $\lceil \frac{d+|C|}{2} \rceil$ where $|C|$ is the number of outputs and d is the number of attributes. The number of messages during learning, $M[\text{BPNN}]_{\text{learn}}$

$$M[\text{BPNN}]_{\text{learn}} = n_{\mathcal{I}} \times \left[\lceil \frac{d+|C|}{2} \rceil (d + |C|) + \lceil \frac{d+|C|}{2} \rceil |C| \right] \times q \quad (2)$$

where $n_{\mathcal{I}}$ is the number of iterations, q is the number of training instances (examples), $\frac{d+|C|}{2}d$ is the number of weights between the input layer and hidden layer, and $\frac{d+|C|}{2}|C|$ is the number of weights between hidden layer and output layer.

The number of messages during recall, $M[\text{BPNN}]_{\text{recall}}$

$$M[\text{BPNN}]_{\text{recall}} = \lceil \frac{d+|C|}{2} \rceil \times (d + |C|) \quad (3)$$

HGN The number of nodes in the HGN is $u \left(\frac{d+1}{2} \right)^2$. Every active node (except top nodes) in the HGN communicates with its u left neighbours, u right neighbours and u nodes at upper layer of the same column. This excepts the edge nodes that are the active nodes at the left end and right end at every layer.

Every edge node (except top nodes) communicates with $2u$ nodes—an edge node at the left end communicates with its u right neighbours and the nodes at the upper layer of the same column, while, an edge node at the right end communicates with

its left neighbours and the nodes at the upper layer of the same column. All of the nodes in the HGN communicate with the base station. The HGN performs a single cycle (one iteration) learning and the communications during learning are equal to the communications during recall.

The number of messages during learning and recall in the HGN are given as follows.

$$\begin{aligned}
 M[\text{HGN}]_{\text{learn/recall}} &= \text{inter-nodes communications} + \text{communications for inputs} + \\
 &\quad \text{communications with a base station for outputs} \\
 &= \sum_{i=2}^{\hbar[\text{HGN}]} \left([(2i-1) - 2] \times 3u + 2 \times 2u \right) + \\
 &\quad u \left(\frac{d+1}{2} \right)^2 + \left(\frac{d+1}{2} \right)^2
 \end{aligned} \tag{4}$$

where $\hbar[\text{HGN}]$ is the height of the structure and $\hbar[\text{HGN}] = \lceil \frac{d+1}{2} \rceil$.

DHGN The DHGN structure consists of several small sub-nets of the HGN where every sub-net processes a subset of attributes. Every sub-net communicates with the base station during learning and recall. The nodes at the base layer of these sub-net sense the inputs while the active top nodes send the sub-networks' outputs to the base station.

Let d_s be the number of attributes processed by a sub-net. The number of nodes in every sub-net is $u \left(\frac{d_s+1}{2} \right)^2$ and the number of sub-nets is $\lceil \frac{d}{d_s} \rceil$. The number of messages during learning and recall in the DHGN are given as follows.

$$\begin{aligned}
 M[\text{DHGN}]_{\text{learn/recall}} &= \text{inter-nodes communications} + \text{communications for inputs} + \\
 &\quad \text{communications with a base station for outputs} \\
 &= \lceil \frac{d}{d_s} \rceil \times \sum_{i=2}^{\hbar[\text{DHGN}]} \left([(2i-1) - 2] \times 3u + 2 \times 2u \right) + \\
 &\quad d \cdot u + \lceil \frac{d}{d_s} \rceil
 \end{aligned} \tag{5}$$

where $\hbar[\text{DHGN}]$ is the height of a sub-net and $\hbar[\text{DHGN}] = \lceil \frac{d_s+1}{2} \rceil$.

P2P-GN During learning, every leaf node in the P2P communicates with a requester node, hence, the number of messages during the learning is $M[\text{P2PGN}]_{learn} = n_H$ where n_H is the number of leaf nodes and it depends on three parameters: d (number of attributes), d_s (segment size) and OV (overlap rate). This is formally given in Equation 3.1 of Chapter 3. However, the number of messages differs between flat recall and convergecast recall.

The number of nodes involved in the convergecast recall depends on n_H and the maximum of connections per peer (m). The smallest possible value of m is equal to 2 and this forms a binary tree. The large m generally reduces the number of messages where the number of messages is equal to the number of leaf nodes if the value m is larger than the number of leaf nodes. The number of messages in the flat recall is given as $M[\text{P2PGN}]_{flat,recall} = 2n_H$, while, the number of messages in the convergecast recall is given in Equation 6 as follows.

$$\begin{aligned} M[\text{P2PGN}]_{convergecast,recall} &\leq \text{communications for inputs} + \text{communications for aggregation} \\ &\leq n_H + (n_R - 1) \end{aligned} \tag{6}$$

where n_R is the total number of nodes of a convergecast tree (refer to Equation 4.1).

C.4 Examples Architectures of the BPNN, HGN, DHGN and P2P-GN

Here, we provide the architectures of BPNN, HGN, DHGN and P2P-GN for comparative analysis in Section 4.2 of Chapter 3. The architectures for these distributed algorithms are constructed for a classification problem with input domain $\{a, b, c\}$ ($u = 3$), a set of 10 outputs (i.e., $\mathcal{C} = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$) and varying numbers of attributes (i.e., $d = \{100, 200, 300, 400, 500, 600, 700\}$). Table C.2 shows the BPNN models in this analysis. The BPNN models have a single layer of hidden neurons where the number of hidden neurons is $\lceil \frac{|\mathcal{C}|+d}{2} \rceil$. Hence, the resulting models consist of three-layer architectures.

Table C.2: The architectures of three-layer BPNN for varying numbers of attributes d and the number of outputs $|\mathcal{C}| = 10$.

Parameters		BPNN (three-layer)
d	$ \mathcal{C} $	
100	10	100-55-10
200	10	200-105-10
300	10	300-155-10
400	10	400-205-10
500	10	500-255-10
600	10	600-305-10
700	10	700-355-10

In Table C.3, we provide the descriptions of the HGN architectures and DHGN architectures for $u = 3$ and with varying numbers of attributes d . The size of input sub-pattern for each DHGN’s sub-net is given as $d_s = \lceil \log_2 d \rceil$.

Table C.3: The architectures of the HGN and the DHGN with varying number of attributes d . For the DHGN, the sub-pattern size is $d_s = \lceil \log_2 d \rceil$.

Parameters			DHGN Architecture		HGN Architecture	
d	d_s	u	Number of sub-networks	Number of nodes	Height	Number of nodes
100	7	3	15	225	50	7,500
200	8	3	29	452	100	30,000
300	9	3	43	681	150	67,500
400	9	3	58	913	250	187,500
500	9	3	72	1,140	300	187,500
600	10	3	86	1,369	301	906,010
700	10	3	100	1,600	351	1,232,010

The HGN has a structure which only depends on parameters d and u , while, the DHGN also depends on another parameter, that is the sub-pattern size, d_s . For simplicity, we decided the value of $d_s = \lceil \log_2 d \rceil$ that is based on the size of d . As shown in Table C.3, the number of nodes in the HGN increases quadratically with an increase in data dimension and the height of the HGN increases linearly. In contrast, the number of nodes in the DHGN increases slowly compared to the HGN.

Then, in Table C.4, we describe the architectures of the P2P-GN with varying numbers of attributes (i.e., $d = \{100, 200, 300, 400, 500, 600, 700\}$). The number of leaf nodes in the P2P-GN depends on parameters d , d_s and overlap rate OV . For

the convergecast recall, the parameter m which defines the number of maximum children per peer is also required.

Table C.4: The P2P-GN architecture with varying numbers of attributes, d . The sub-pattern size $d_s = \lceil \log_2 d \rceil$ and $OV = \lceil \frac{d_s}{2} \rceil$. The parameter m (the number of maximum connection per node) is set to 10 in convergecast recall.

Parameters				P2P-GN Architecture		
d	d_s	m	OV	Learning/flat structure	Convergecast tree	
				Number of leaf nodes	Number of nodes	Height
100	7	10	4	32	37	2
200	8	10	4	39	55	2
300	9	10	5	74	83	2
400	9	10	5	99	110	2
500	9	10	5	124	155	3
600	10	10	5	119	150	3
700	10	10	5	139	170	3

The number of leaf nodes is smaller when the number of attributes is 600 than when it is 500 since we set $d_s = \lceil \log_2 500 \rceil = 9$ and $OV = \lceil \frac{9}{2} \rceil = 5$ when the number of attributes is 500; and $d_s = \lceil \log_2 600 \rceil = 10$ and $OV = \lceil \frac{10}{2} \rceil = 5$ when the data dimension is 600. Hence, this results in the number of leaf nodes at $d = 500$ to be 124, while, at $d = 600$ to be 119.

Appendix D: DASMET

D.5 Examples of DASMET Recall Procedure

An example of DASMET recall procedure is given here through illustrative examples in Figures D.1, D.2 and D.3. Given that three is the height of the DASMET structure

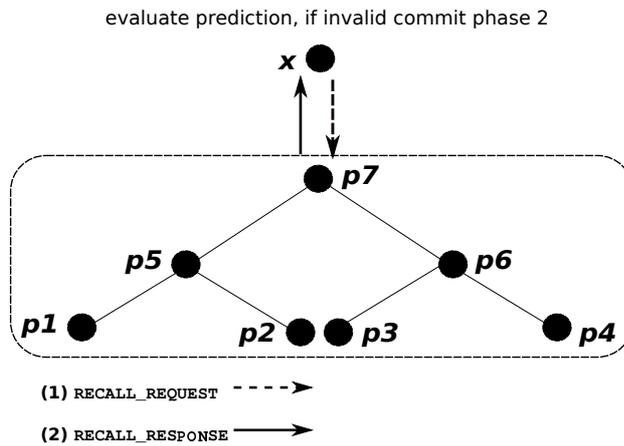


Figure D.1: An example of DASMET recall phase 1 where a peer x sends a recall request to the peer $p7$ and $p7$ then responds with the local prediction. If the termination condition C_0 is not satisfied then the peer x commits phase 2 as shown in Figure D.2.

in this example, the number of phases n_{phases} is $0 < n_{phases} \leq 3$. Upon receiving the response from $p1$, a peer x validates the prediction against Rule 5.5 and Rule 5.4 of Chapter 5. If the termination condition C_0 (by Rule 5.4 of Chapter 5) is not satisfied, then it sends the recall requests to the lower level ($p2$ and $p3$) at phase 2 (see Figure D.2). The receiving peers $p2$ and $p3$ then process these recall requests in parallel before sending the local prediction back to the peer x . Peer x then aggregates the received local predictions and produces a single prediction which is

again tested against Rule 5.5 and Rule 5.4. This iterative process continues to phase 3, provided that Rule 5.4 is not reached (see Figure D.3). After Rule 5.4 has been satisfied, the final decision is made by function `finalPrediction(.)`.

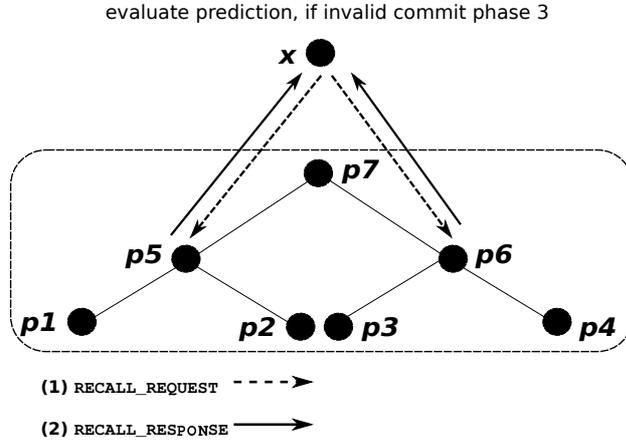


Figure D.2: In the phase 2, the peer x sends recall requests ψ_5 and ψ_6 to the peer p_5 and p_6 , respectively. The responses from p_5 and p_6 are then aggregated into a single prediction which then is tested against the termination condition C_0 . In case the condition C_0 is still unfulfilled, then the peer x commits phase 3.

Here, the recall is a multi-phase operation which can be stopped at any level, once a termination condition Rule 5.4 has been satisfied. The worst-case happens when given a tree with \bar{h} -height; the number of phases that is required before termination is \bar{h} . However, an optimal recall with respect to recall time and communication efficiency is achieved when a valid prediction can be obtained after querying from the root node (the number of phases that is required before termination is 1).

D.6 Dataset Factor to The Probability of Optimal Recall Prediction

The following Proposition D.1 states the effect of data dimension (number of attributes) on the probability of an optimal recall prediction.

Proposition D.1. *Given a dataset A with features \mathbf{a} is duplicated into two datasets, B and C . Given \mathbf{b} is a subset of features that are selected from \mathbf{a} to represent dataset*

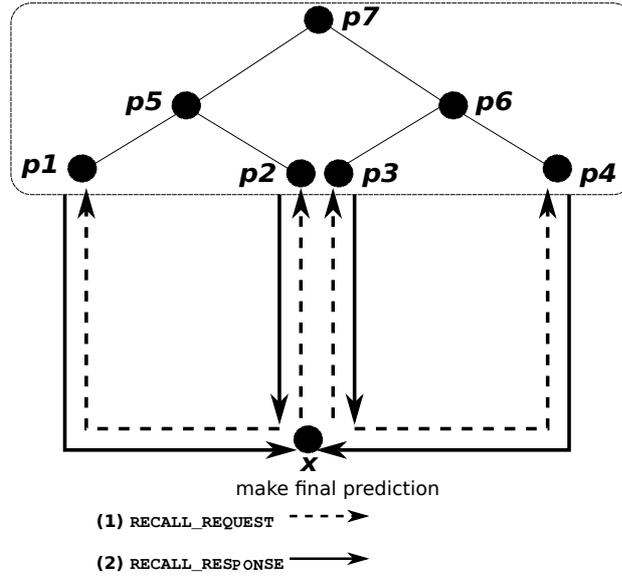


Figure D.3: In the phase 3, the recall requests are submitted to ψ_1, ψ_2, ψ_3 and ψ_4 and local predictions from p_1, p_2, p_3 and p_4 are aggregated at the peer x . The final decision is made at this stage as the condition C_0 is reached when the number of phases is equal to n_{phase} .

B while \mathbf{c} is a subset of features that are selected from \mathbf{a} to represent dataset C where $\mathbf{b}, \mathbf{c} \in \mathbf{a}$. $P(\widehat{OR}[B]) < P(\widehat{OR}[C])$ during recall when the size of \mathbf{b} is **greater than** \mathbf{c} where $P(\widehat{OR}[y])$ is the probability of an optimal recall prediction in dataset y .

Proof. For example, given $d(B) = 2$ and $d(C) = 3$ where $d(y)$ is the data dimension for dataset y . The probability that a pattern j is found at a root node is $\frac{1}{2^2}$ in dataset B and the probability that a pattern of j is found at a root node in dataset C is $\frac{1}{2^3}$, hence $P(\widehat{OR}[B]) < P(\widehat{OR}[C])$. \square

The following Proposition D.2 states an effect of training data size on the probability of an optimal recall time.

Proposition D.2. Given two datasets of the same domain, B and C . $P(\widehat{OR}[B]) > P(\widehat{OR}[C])$ during recall when the training size in B is larger than the training size in C where $P(\widehat{OR}[y])$ is the probability of an optimal recall prediction in dataset y .

Proof. For example, given two datasets B and C with a classification domain of $\{0, 1\}^5$. Let the size of training data in B be 24 and the size of training data in C

be 4. Hence, the probability of optimal recall prediction in dataset B is $\frac{24}{2^5}$ which is higher than $\frac{4}{2^5}$ in dataset C , or $P(\widehat{OR}[B]) > P(\widehat{OR}[C])$. \square

References

- [1] TinEye Reverse Image Search, April 2011. [Online viewed on April 7, 2011]
<http://tineye.com/>.
- [2] V. Aggarwal, O. Akonjang, A. Feldmann, R. Tashev, and S. Mohrs. Reflecting P2P User Behaviour Models in a Simulation Environment. In *Proceedings of the 16th Euromicro Conference on Parallel, Distributed and Network-Based Processing , (PDP 2008), Toulouse, France, 13 - 15 February, 2008*, pages 516–523, Washington, DC, USA, 2008. IEEE Computer Society.
- [3] D. Agrawal, S. Das, and A. El Abbadi. Big Data and Cloud Computing: Current State and Future Opportunities. In *Proceedings of the 14th International Conference on Extending Database Technology, (EDBT/ICDT '11), Uppsala, Sweden, 22 - 24 March, 2011*, pages 530–533, New York, NY, USA, 2011. ACM.
- [4] B. Al-Duwairi, I. Khater, and O. Al-Jarrah. Detecting image spam using image texture features. *International Journal for Information Security Research (IJISR)*, 2(3/4):344–353, 2012.
- [5] A. Amir, A. H. M. Amin, and B. Srinivasan. P2P-Based Image Recognition for Component Tracking in a Large Engineering Domain. In P. Ivnyi and B. Topping, editors, *Proceedings of the Second International Conference on Parallel, Distributed, Grid and Cloud Computing for Engineering, Corsica, 12 - 15 April, France, 2011*, Stirlingshire, UK, 2011. Civil-Comp Press.

-
- [6] S. Androutsellis-Theotokis and D. Spinellis. A survey of peer-to-peer content distribution technologies. *ACM Computing Survey*, 36:335–371, December 2004.
- [7] H. H. Ang, V. Gopalkrishna, S. Hoi., and W. K. Ng. Cascade RSVM in Peer-to-Peer Networks. 5211:55–70, 2008.
- [8] H. H. Ang, V. Gopalkrishnan, S. Hoi, W. K. Ng, and A. Datta. Classification in P2P Networks by Bagging Cascade RSVMs. In *Proceeding of Sixth International Workshops Databases, Information Systems, and Peer-to-Peer Computing, (DBISP2P 2008), Auckland, New Zealand, 23 - 23 August, 2008*, pages 13–25, Auckland, New Zealand, August 2008. DBLP.
- [9] H. H. Ang, V. Gopalkrishnan, S. C. H. Hoi, and W. K. Ng. Adaptive ensemble classification in P2P networks. In *Proceedings of the 15th International Conference on Database Systems for Advanced Applications, (DAS-FAA'10), Tsukuba, Japan, 1-4 April 2010*, pages 34–48, Berlin, Heidelberg, 2010. Springer-Verlag.
- [10] AudibleMagic. *AudibleMagic Software*. April 2012.
<http://audiblemagic.com/>.
- [11] B. Babcock and C. Olston. Distributed Top-k Monitoring. In *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data/Principles of Database Systems, (SIGMOD '03), San Diego, California, 9-12 June, 2003*, pages 28–39, San Diego, California, 2003. ACM.
- [12] S. Baccianella, A. Esuli, and F. Sebastiani. Feature selection for ordinal text classification. *Neural Computing*, 26(3):557–591, Mar. 2014.
- [13] K. Bache and M. Lichman. UCI machine learning repository, 2013.
<http://archive.ics.uci.edu/ml>.

-
- [14] E. Bauer and R. Kohavi. An Empirical Comparison of Voting Classification Algorithms: Bagging, Boosting, and Variants. *Machine Learning*, 36(1-2):105–139, July 1999.
- [15] M. Bawa, A. Gionis, H. Garcia-Molina, and R. Motwani. The Price of Validity in Dynamic Networks. *Journal of Computer and System Sciences*, 73(3):245–264, 2007.
- [16] Y. Bengio. *Neural Networks for Speech and Sequence Recognition*. International Thomson Computer Press, University of Michigan, Nov 2007.
- [17] K. Bhaduri, R. Wolff, C. Giannella, and H. Kargupta. Distributed Decision-Tree Induction in Peer-to-Peer Systems. *Statistical Analysis and Data Mining*, 1:85–103, June 2008.
- [18] BitTorrent. Bittorrent website, January 2012. [Online viewed Nov 1, 2013] <http://www.bittorrent.com/>.
- [19] E. Blanzieri and A. Bryl. A Survey of Learning-based Techniques of Email Spam Filtering. *Artificial Intelligence Review*, 29(1):63–92, Mar. 2008.
- [20] C. Blum. Beam-ACO: Hybridizing Ant Colony Optimization with Beam Search: An Application to Open Shop Scheduling. *Computer Operating Research*, 32(6):1565–1591, June 2005.
- [21] R. R. Bouckaert, E. Frank, M. Hall, R. Kirkby, P. Reutemann, A. Seewald, and D. Scuse. WEKA Manual for Version 3-7-11, April 2014. <http://www.cs.waikato.ac.nz/ml/weka/documentation.html>.
- [22] S. Boyd, A. Ghosh, B. Prabhakar, and D. Shah. Randomized gossip algorithms. *IEEE Transactions on Information Theory*, 52(6):2508–2530, 2006.
- [23] L. Breiman. Bagging Predictors. *Machine Learning*, 24(2):123–140, 1996.
- [24] L. Breiman. Pasting Small Votes for Classification in Large Databases and On-Line. *Machine Learning*, 36:85–103, July 1999.

-
- [25] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. *Classification and Regression Trees*. Statistics/Probability Series. Wadsworth Publishing Company, Belmont, California, U.S.A., 1984.
- [26] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. *Classification and Regression Trees*. Statistics/Probability Series. Wadsworth Publishing Company, Belmont, California, U.S.A., 1984.
- [27] M. Browne, S. S. Ghidary, and N. M. Mayer. Convolutional neural networks for image processing with applications in mobile robotics. In B. Prasad and S. Prasanna, editors, *Speech, Audio, Image and Biomedical Signal Processing using Neural Networks*, pages 327–349. Springer Berlin Heidelberg, 2008.
- [28] BuddyBackup. Buddybackup website, April 2012. [Online viewed April 6, 2013]
<http://www.buddybackup.com>.
- [29] C. J. C. Burges. A Tutorial on Support Vector Machines for Pattern Recognition. *Data Mining Knowledge Discovery*, 2(2):121–167, June 1998.
- [30] A. Byron. FTC sounds alarm: data leaking onto P2P networks, February 2010. [Online; posted on November 6, 2013]
<http://content.usatoday.com/communities/technologylive/post/2010/02/ftc-sounds-alarm-about-data-leaked-onto-p2p-networks/>.
- [31] M. Cai. *Distributed Aggregation Schemes for Scalable Peer-to-Peer and Grid Computing*. PhD thesis, Los Angeles, CA, USA, 2006.
- [32] D. Caragea, A. Silvescu, and V. Honavar. A framework for learning from distributed data using sufficient statistics and its application to learning decision trees. *International Journal of Hybrid Intelligent Systems*, 1:2004, 2004.
- [33] B. Carlsson and R. Gustavsson. The Rise and Fall of Napster - An Evolutionary Approach. In J. Liu, C. Y. Pong, H. L. Chun, K.-Y. J. Ng, and T. Ishida,

-
- editors, *Active Media Technology, 6th International Computer Science Conference, AMT 2001, Hong Kong, China, December 18-20, 2001, Proceedings*, volume 2252 of *Lecture Notes in Computer Science*, pages 347–354. Springer, 2001.
- [34] P. K. Chan and S. J. Stolfo. Learning Arbiter and Combiner Trees from Partitioned Data for Scaling Machine Learning. In *Proceedings of the First International Conference on Knowledge Discovery and Data Mining, Montreal, Quebec, Canada, 2021 August, 1995*, pages 39–44. AAAI Press, 1995.
- [35] Y. W. Chan, T. H. Ho, P. Shih, and Y. Chung. Malugo: A Peer-to-Peer Storage System. *International Journal of Ad Hoc Ubiquitous Computing*, 5: 209–218, May 2010.
- [36] S. A. Chatzichristofis and Y. S. Boutalis. FCTH: Fuzzy Color and Texture Histogram - A Low Level Feature for Accurate Image Retrieval. In *Proceedings of the 2008 Ninth International Workshop on Image Analysis for Multimedia Interactive Services, (WIAMIS '08), Klagenfurt, Austria, 7-9 May, 2008*, WIAMIS '08, pages 191–196, Washington, DC, USA, 2008. IEEE Computer Society.
- [37] S. A. Chatzichristofis and Y. S. Boutalis. CEDD: Color and Edge Directivity Descriptor: A Compact Descriptor for Image Indexing and Retrieval. In A. Gasteratos, M. Vincze, and J. Tsotsos, editors, *Computer Vision Systems*, volume 5008 of *Lecture Notes in Computer Science*, pages 312–322. Springer Berlin Heidelberg, 2008.
- [38] X. Cheng, J. Xu, J. Pei, and J. Liu. Hierarchical Distributed Data Classification in Wireless Sensor Networks. *Comput. Commun.*, 33(12):1404–1413, July 2010.
- [39] P.-A. Chirita, J. Diederich, and W. Nejdl. MailRank: Using Ranking for Spam Detection. In *Proceedings of the 14th ACM International Conference on*

-
- Information and Knowledge Management*, (CIKM '05), Bremen, Germany, 31 October - 05 November, 2005, CIKM '05, pages 373–380, New York, NY, USA, 2005. ACM.
- [40] N. Christin, A. S. Weigend, and J. Chuang. Content Availability, Pollution and Poisoning in File Sharing Peer-to-Peer networks. In *Proceedings of the 6th ACM conference on Electronic Commerce*, (EC '05), Vancouver, Canada, 5-8 June, 2005, pages 68–77, New York, NY, USA, 2005. ACM.
- [41] C. T. Chu, S. K. Kim, Y. A. Lin, Y. Yu, G. R. Bradski, A. Y. Ng, and K. Olukotun. Map-Reduce for Machine Learning on Multicore. In B. Schölkopf, J. C. Platt, and T. Hoffman, editors, *NIPS*, pages 281–288. MIT Press, 2006.
- [42] clip2. The Gnutella Protocol Specification v0.4 (Document Revision 1.2), 2001. [Online viewed 4 Jan 2014]
<http://www.clip2.com/GnutellaProtocol104.pdf>.
- [43] J. Considine, F. Li, G. Kollios, and J. Byers. Approximate Aggregation Techniques for Sensor Databases. In *Proceedings 20th International Conference on Data Engineering, (ICDE 2004)*, Boston, USA, 30 March - 2 April, 2004, pages 449–460, Boston, MA, USA, March 2004.
- [44] F. Cornelli, E. Damiani, S. D. C. di Vimercati, S. Paraboschi, and P. Samarati. Choosing Reputable Servents in a P2P Network. In *Proceedings of the 11th International Conference on World Wide Web, (WWW '02)*, 7-11 May, 2002, Honolulu, Hawaii, WWW '02, pages 376–386, New York, NY, USA, 2002. ACM.
- [45] Y. M. G. Costa, L. S. Oliveira, A. L. Koerich, F. Gouyon, and J. G. Martins. Music Genre Classification using LBP Textural Features. *Signal Processing*, 92(11):2723–2737, Nov. 2012.

-
- [46] G. Costantini, D. Casali, and R. Perfetti. Neural Associative Memory Storing Gray-coded Gray-scale Images. *IEEE Transactions on Neural Networks*, 14(3):703 – 707, May 2003.
- [47] L. F. Cranor and B. A. LaMacchia. Spam! *Communication ACM*, 41(8):74–83, Aug. 1998.
- [48] CrashPlan. Crashplan website, April 2012. [Online viewed April 6, 2012] <http://www.crashplan.com>.
- [49] A. Crespo and H. Garcia-Molina. Semantic overlay networks for p2p systems. In *Proceedings of the Third International Conference on Agents and Peer-to-Peer Computing, (AP2PC'04), New York, USA, 19 July, 2004*, AP2PC'04, pages 1–13, Berlin, Heidelberg, 2005. Springer-Verlag.
- [50] D. N. da Hora, D. F. Macedo, L. B. Oliveira, I. G. Siqueira, A. A. F. Loureiro, J. M. Nogueira, and G. Pujolle. Enhancing peer-to-peer content discovery techniques over mobile ad hoc networks. *Comput. Commun.*, 32(13-14):1445–1459, Aug. 2009.
- [51] E. Damiani, D. C. di Vimercati, S. Paraboschi, P. Samarati, and F. Violante. A Reputation-based Approach for Choosing Reliable Resources in Peer-to-Peer Networks. In *CCS '02: Proceedings of the 9th ACM conference on Computer and communications security*, pages 207–216, New York, NY, USA, 2002. ACM.
- [52] E. Damiani, S. D. C. di Vimercati, S. Paraboschi, and P. Samarati. P2P-Based Collaborative Spam Detection and Filtering. In *Proceedings of the Fourth International Conference on Peer-to-Peer Computing, (P2P '04), Zurich, Switzerland, 15-17 August 2004*, pages 176–183, Washington, DC, USA, August 2004. IEEE Computer Society.
- [53] K. Das, K. Bhaduri, K. Liu, and H. Kargupta. Distributed Identification of Top-1 Inner Product Elements and its Application in a Peer-to-Peer Network.

-
- IEEE Transaction on Knowledge and Data Engineering*, 20:475–488, April 2008.
- [54] S. Datta, K. Bhaduri, C. Giannella, R. Wolff, and H. Kargupta. Distributed Data Mining in Peer-to-Peer Networks. *Internet Computing, IEEE*, 10(4): 18–26, 2006.
- [55] J. Dean and S. Ghemawat. MapReduce: simplified data processing on large clusters. *Communication ACM*, 51:107–113, 2008.
- [56] P. Dhungel, X. Hei, K. W. Ross, and N. Saxena. The Pollution Attack in P2P Live Video Streaming: Measurement Results and Defenses. In *Proceedings of the 2007 workshop on Peer-to-Peer Streaming and IP-TV, (P2P-TV '07), Kyoto, Japan, 27-31 August, 2007*, P2P-TV '07, pages 323–328, Kyoto, Japan, 2007. ACM.
- [57] J. Dinger and H. Hartenstein. Defending the Sybil Attack in P2P Networks: Taxonomy, Challenges, and a Proposal for Self-Registration. In *In ARES'06: Proceedings of the First International Conference on Availability, Reliability and Security, (ARES'06), Vienna, Austria, 20 - 22 April, 2006*, pages 756–763, Vienna, Austria, April 2006.
- [58] Donkeyfakes. Donkeyfakes homepage, June 2013. [Online viewed June 6, 2013] <http://www.aboutus.org/DonkeyFakes.net>.
- [59] C. Douligieris and A. Mitrokotsa. DDoS Attacks and Defense Mechanisms: Classification and State-of-the-art. *Computer Network*, 44(5):643–666, Apr. 2004.
- [60] M. Dredze, R. Gevartyahu, and A. Elias-Bachrach. Learning fast classifiers for image spam. In *Fourth Conference on Email and Anti-Spam, (CEAS 2007), Mountain View, California, 2-3 August, 2007*, 2007.

-
- [61] P. Druschel and A. Rowstron. PAST: A Large-scale, Persistent Peer-to-Peer Storage Utility. In *Proceedings of the Eighth Workshop on Hot Topics in Operating Systems, Hot Topics in Operating Systems, Elmau/Oberbayern, Germany, 20-23 May, 2001*, pages 75–80, Elmau, Germany, May 2001. IEEE Computer Society.
- [62] R. O. Duda, P. E. Hart, and D. G. Stork. *Pattern Classification*. Wiley & Sons, Inc., New York, NY, USA, 2nd edition, 2001.
- [63] eMule. emule software(v0.50a), January 2010. [Online viewed on Nov 2, 2013] <http://www.emule-project.net/>.
- [64] M. Fanty and R. Cole. Spoken Letter Recognition. In *Proceedings of the Conference on Advances in Neural Information Processing Systems 3, (NIPS-3), Colorado, United States, 26 - 29 November, 1990*, NIPS-3, pages 220–226, San Francisco, CA, USA, 1990. Morgan Kaufmann Publishers Inc.
- [65] P. A. Flach. The geometry of ROC space: understanding machine learning metrics through ROC isometrics. In *Proceedings of the 20th International Conference on Machine Learning, Washington, (ICML-03), DC, 21-24 August, 2003*, Washington, DC, August 2003. AAAI Press.
- [66] P. A. Forero, A. Cano, and G. B. Giannakis. Consensus-Based Distributed Support Vector Machines. *Machine Learning Research*, 99:1663–1707, August 2010.
- [67] Y. Freund, R. Iyer, R. E. Schapire, and Y. Singer. An Efficient Boosting Algorithm for Combining Preferences. In *Machine Learning Research*, pages 170–178. Morgan Kaufmann, 1998.
- [68] J.-M. Geusebroek, G. J. Burghouts, and A. W. M. Smeulders. The Amsterdam library of object images. *International Journal of Computer Vision*, 61(1): 103–122, 2005.

- [69] A. Ghodsi, L. O. Alima, and S. Haridi. Symmetric Replication for Structured Peer-to-Peer Systems. In *Proceedings of the 2005/2006 International Conference on Databases, Information Systems, and Peer-to-peer Computing, (DBISP2P'05/06), Trondheim, Norway, 28-29 August, 2005*, DBISP2P'05/06, pages 74–85, Berlin, Heidelberg, 2005. Springer-Verlag.
- [70] L. Gitte. Spam text messages a growing problem, July 2012. [Online posted on June 16, 2012]
<http://www.jsonline.com/news/wisconsin/spam-text-messages-a-growing-problem-6d62f17-162666806.html>.
- [71] Google. Google Chat Website, April 2013. [Online viewed on April 7, 2013]
<http://www.google.com/talk/>.
- [72] M. Gorman. Qualcomm's AllJoyn P2P software framework adds audio streaming and notifications, we go eyes-on, February 2013. [Online viewed on January 10, 2014]
<http://www.engadget.com/2013/02/25/qualcomm-alljoyn-p2p-software-framework-audio-streaming-eyes-on/>.
- [73] Q. Gu, K. Bai, H. Wang, P. Liu, and C.-H. Chu. Modeling of Pollution in P2P File Sharing Systems. In *Proceedings of 3rd IEEE Consumer Communications and Networking Conference, (CCNC 2006), Las Vegas, Nevada, USA, 8 - 10 January, 2006*, pages 1033–1037, January 2006.
- [74] I. Gupta, R. v. Renesse, and K. P. Birman. Scalable Fault-Tolerant Aggregation in Large Process Groups. In *Proceedings of the 2001 International Conference on Dependable Systems and Networks (formerly: FTCS), (DSN'01), Gteborg, Sweden, 30 June - 4 July, 2001*, DSN '01, pages 433–442, Washington, DC, USA, 2001. IEEE Computer Society.

-
- [75] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten. The WEKA data mining software: an update. *SIGKDD Explorer Newsletters*, 11(1):10–18, Nov. 2009.
- [76] M. A. Hall. Correlation-based Feature Selection for Discrete and Numeric Class Machine Learning. In *Proceedings of the Seventeenth International Conference on Machine Learning, (ICML '00), Stanford, CA, USA, 29 June - 2 July, 2000*, ICML '00, pages 359–366, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc.
- [77] G. Hamerly and C. Elkan. Learning the K in K-means. In *Proceedings of the 17th Annual Conference on Neural Information Processing Systems, (NIPS 2003), Vancouver, Canada, 8-13 December, 2003*, pages 1–8, Nevada, USA, Dec 2003. MIT Press.
- [78] S. C. Han and Y. Xia. Optimal Leader Election Scheme for Peer-to-Peer Applications. In *Proceedings of the Sixth International Conference on Networking, (ICN '07), Sainte-Luce, Martinique, France, 22-28 April 2007*, ICN '07, page 29, Washington, DC, USA, 2007. IEEE Computer Society.
- [79] R. Hanka and T. P. Harte. Curse of Dimensionality: Classifying Large Multi-Dimensional Images with Neural Networks. In M. Krn and K. Warwick, editors, *Computer Intensive Methods in Control and Signal Processing*, pages 249–260. Birkhuser Boston, 1997.
- [80] R. Haralick, K. Shanmugam, and I. Dinstein. Textural features for image classification. *Systems, Man and Cybernetics, IEEE Transactions on*, 3(6): 610–621, 1973.
- [81] J. A. Hartigan and M. A. Wong. A K-means clustering algorithm. *Applied Statistics*, 28:100–108, 1979.
- [82] J. Hartley. Peer-to-Peer Networking: A Mobile Coming of Age, July 2012. [Online viewed on January 11, 2013]

[http://software.intel.com/en-us/articles/
peer-to-peer-networking-a-mobile-coming-of-age](http://software.intel.com/en-us/articles/peer-to-peer-networking-a-mobile-coming-of-age).

- [83] O. Heckmann, A. Bock, A. Mauthe, and R. Steinmetz. The eDonkey File-sharing Network. In *Proceedings of the Workshop on Algorithms and Protocols for Efficient Peer-to-Peer Applications, (INFORMATIK'04), Ulm, Germany, 23 September, 2004*, pages 2–6, Sept 2004.
- [84] T. K. Ho. The Random Subspace Method for Constructing Decision Forests. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20:832–844, 1998.
- [85] J. J. Hopfield. Neural Networks and Physical Systems with Emergent Collective Computational Abilities. In J. A. Anderson and E. Rosenfeld, editors, *Neurocomputing: Foundations of Research*, pages 457–464. MIT Press, Cambridge, MA, USA, 1988.
- [86] C. Hua Li and J. Xiangji Huang. Spam Filtering using Semantic Similarity Approach and Adaptive BPNN. *Neurocomputing*, 92:88–97, Sept. 2012.
- [87] Q. Iqbal and J. K. Aggarwal. CIRES: A System for Content-Based Retrieval in Digital Image Libraries. In *Proceedings of International Conference on Control, Automation, Robotics and Vision, (ICARCV 2002), Singapore, 2-5 December, 2002*, pages 205–210, Dec. 2002.
- [88] M. Jelasity, A. Montresor, and O. Babaoglu. Gossip-based aggregation in large dynamic networks. *ACM Transactions on Computer Systems*, 23(3):219–252, Aug. 2005.
- [89] JFeatureLib. JFeatureLib: A free java library containing feature descriptors and detectors, April 2013. [Online viewed on April 6, 2013]
<http://code.google.com/p/jfeaturelib/>.
- [90] D. Jia. Cost-effective Spam Detection in P2P File-Sharing Systems. In *Proceedings of the 2008 ACM workshop on Large-Scale Distributed Systems for*

-
- Information Retrieval, (LSDS-IR '08), Napa Valley, California, USA, 27 -29 October, 2008*, LSDS-IR '08, pages 19–26. ACM, 2008.
- [91] J.-R. Jiang, C.-T. King, and C.-H. Liao. MUREX: A mutable replica control scheme for structured Peer-to-Peer storage systems. In Y.-C. Chung and J. Moreira, editors, *Advances in Grid and Pervasive Computing*, volume 3947 of *Lecture Notes in Computer Science*, pages 93–102. Springer Berlin Heidelberg, 2006.
- [92] X. Jin and S.-H. G. Chan. Detecting Malicious Nodes in Peer-to-Peer Streaming by Peer-based Monitoring. *ACM Transaction Multimedia Computing Communication Application*, 6(2):9:1–9:18, Mar. 2010.
- [93] V. John and E. Trucco. Charting-based subspace learning for video-based human action classification. *Mach. Vision Appl.*, 25(1):119–132, January 2014.
- [94] S. D. Kamvar, M. T. Schlosser, and H. Garcia-Molina. The Eigentrust Algorithm for Reputation Management in P2P networks. In *Proceedings of the 12th International Conference on World Wide Web, (WWW '03), Budapest, Hungary, 20-24 May, 2003*, WWW '03, pages 640–651, New York, NY, USA, 2003. ACM.
- [95] R. Kapelko. Towards fault-tolerant chord p2p system: Analysis of some replication strategies. In Y. Ishikawa, J. Li, W. Wang, R. Zhang, and W. Zhang, editors, *Web Technologies and Applications*, volume 7808 of *Lecture Notes in Computer Science*, pages 686–696. Springer Berlin Heidelberg, 2013.
- [96] Kazaa. Kazaa website, April 2011. [Online viewed on 2 January, 2013] <http://www.kazaa.com/>.
- [97] D. Kempe, A. Dobra, and J. Gehrke. Gossip-based computation of aggregate information. In *Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science, FOCS '03*, pages 482–, Washington, DC, USA, 2003. IEEE Computer Society.

- [98] A. Khan and R. Heckel. Evaluating super node selection and load balancing in p2p voip networks using stochastic graph transformation. In M. S. Obaidat, J. Sevillano, and J. Filipe, editors, *E-Business and Telecommunications*, volume 314 of *Communications in Computer and Information Science*, pages 60–73. Springer Berlin Heidelberg, 2012.
- [99] A. I. Khan. A Peer-to-Peer Associative Memory for Intelligent Information Systems. In *Proceeding Thirteenth Australasian Conference on Information Systems, (ACIS 2002), Melbourne, Australia, December, 2002*, 2002.
- [100] A. I. Khan and P. Mihailescu. Parallel Pattern Recognition Computations within a Wireless Sensor Network. In *Proceedings of 17th International Conference on the Pattern Recognition, (ICPR'04), Cambridge, UK, 23-26 August, 2004*, volume 1, pages 777–780, Washington, DC, USA, 2004. IEEE Computer Society.
- [101] A. I. Khan, M. Baqer, and Z. A. Baig. Implementing a graph neuron array for pattern recognition within unstructured wireless sensor networks. *Lecture Notes in Computer Science*, pages 208–217, 2006.
- [102] A. Kingsley-Hughes. Now Skype Users Have Voicemail Spam Malware To Worry About, October 2012. [Online posted on October 10, 2012]
<http://www.forbes.com/sites/adriankingsleyhughes/2012/10/10/now-skype-users-have-voicemail-spam-malware-to-worry-about/>.
- [103] T. Kohonen, M. R. Schroeder, and T. S. Huang, editors. *Self-Organizing Maps*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 3rd edition, 2001.
- [104] G. K. Konstantinos Trohidis, Grigorios Tsoumakas and I. Vlahavas. Multi-label classification of music by emotion. *EURASIP Journal on Audio, Speech, and Music Processing 2011*, 4, 2011.
- [105] K. B. Korb and A. E. Nicholson. *Bayesian Artificial Intelligence, Second Edition*. CRC Press, Inc., Boca Raton, FL, USA, 2nd edition, 2010.

-
- [106] B. Kosko. Bidirectional associative memories. *IEEE Transactions on Systems, Man and Cybernetics, Part B*, 30:59–68, 2006.
- [107] W. Kowalczyk, M. Jelasity, and A. E. Eiben. Towards Data Mining in Large and Fully Distributed Peer-To-Peer Overlay Networks. In *Proceedings of Belgium-Netherlands Conference on Artificial Intelligence, (BNAIC'03), Delft, The Netherlands, 7 - 8 November, 2003*, pages 203–210, Oct. 2003.
- [108] A. Lazarevic and Z. Obradovic. Boosting algorithms for parallel and distributed learning. *Distributed Parallel Databases*, 11(2):203–229, Mar. 2002.
- [109] Y. LeCun and C. Cortes. MNIST handwritten digit database, 2010. [Online viewed on June 5, 2012]
<http://yann.lecun.com/exdb/mnist/>.
- [110] Y. Lee and O. Mangasarian. RSVM: Reduced Support Vector Machines. In *Proceedings of the First SIAM International Conference on Data Mining, Chicago, USA, 5 - 7 April 2001*, pages 00–07, Chicago,IL,USA, April 2001. SIAM Philadelphia.
- [111] K. Leibnitz, T. Hofeld, N. Wakamiya, and M. Murata. On Pollution in eDonkey-like Peer-to-Peer File-sharing Networks. In *Proceedings 13th GI/ITG Conference on Measuring, Modelling and Evaluation of Computer and Communication Systems, (MMB 2006,) Nürnberg, Germany, 27-29 March, 2006*, pages 1–18, Nurnberg, Germany, March 2006.
- [112] B. Li, T. W. Chow, and D. Huang. A novel feature selection method and its application. *Journal of Intelligent Information System*, 41(2):235–268, Oct. 2013.
- [113] J. Liang, R. Kumar, Y. Xi, and K. W. Ross. Pollution in P2P File Sharing Systems. In *Proceedings IEEE 24th International Conference on Computer Communications, (INFOCOM 2005), Miami FL, USA, 13 - 15 March, 2005*, pages 1174–1185, Miami , FL,USA, March 2005. IEEE Society.

-
- [114] J. Liang, R. Kumar, and K. W. Ross. The FastTrack overlay: a measurement study. *Computer Network*, 50(6):842–858, 2006.
- [115] J. Liang, N. Naoumov, and K. W. Ross. The Index Poisoning Attack in P2P File Sharing System. In *Proceedings 25th IEEE International Conference on Computer Communications, (INFOCOM 2006), Barcelona, Spain, 23 - 29 April, 2006*, pages 1–12, April 2006.
- [116] T. Liang and Y. Pi. On Spam Detection Based on Cognitive Pattern Recognition. In *International Conference on Computational Intelligence and Security Workshops, (CIS Workshops 2007), Harbin, Heilongjiang, China, 15-19 December, 2007*, pages 136–139, 2007.
- [117] E. Limer. BitTorrent’s P2P Streaming Takes the Lag Out of Live Video, December 2013. [Online viewed on January 11, 2014]
<http://gizmodo.com/5990014/bittorrents-p2p-streaming-protocol-is-here-to-change-web-video-forever>.
- [118] Limewire. Limewire website, January 2010. [Online viewed on Jan 9, 2013]
<http://www.limewire.com/>.
- [119] K.-M. Lin and C.-J. Lin. A study on Reduced Support Vector Machines. *IEEE Transactions on Neural Networks*, 14(6):1449–1459, 2003.
- [120] P.-C. Lin, M.-D. Liu, Y.-D. Lin, and Y.-C. Lai. Accelerating Web Content Filtering by the Early Decision Algorithm. *IEICE - Transaction Information System*, E91-D(2):251–257, Feb. 2008.
- [121] Y. Liu, Y. Jiang, and J. Yang. Feature reduction with inconsistency. *International Journal of Cognitive Informatics and Natural Intelligence*, 4(2):77–87, Apr. 2010.

-
- [122] V. Lo, D. Zhou, Y. Liu, C. GauthierDickey, and J. Li. Scalable supernode selection in peer-to-peer overlay networks. In *Proceedings of the Second International Workshop on Hot Topics in Peer-to-Peer Systems, (HOT-P2P '05), La Jolla, USA, 21 July, 2005*, pages 18–27, 2005.
- [123] V. M. Lo, D. Zappala, D. Zhou, Y. Liu, and S. Zhao. Cluster Computing on the Fly: P2P Scheduling of Idle Cycles in the Internet. In *Proceedings of the Third International Conference on Peer-to-Peer Systems, (IPTPS'04), La Jolla, CA, 26-27 February, 2004*, pages 227–236, July 2004.
- [124] E. K. Lua, J. Crowcroft, M. Pias, R. Sharma, and S. Lim. A Survey and Comparison of Peer-to-Peer Overlay Network Schemes. *IEEE Communications Surveys and Tutorials*, 7:72–93, 2005.
- [125] P. Luo, H. Xiong, K. Lü, and Z. Shi. Distributed classification in Peer-to-Peer networks. In *Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, (KDD '07), San Jose, California, USA, 12 - 15 August, 2007*, pages 968–976, 2007.
- [126] S. Madden, M. Franklin, J. Hellerstein, and W. Hong. TinyDB: an acquisitional query processing system for sensor networks. *ACM Transactions on Database Systems (TODS)*, 30(1):122–173, 2005.
- [127] S. Maldonado and G. L'Huillier. SVM-based feature selection and classification for email filtering. In P. Latorre Carmona, J. S. Snchez, and A. L. Fred, editors, *Pattern Recognition - Applications and Methods*, volume 204 of *Advances in Intelligent Systems and Computing*, pages 135–148. Springer Berlin Heidelberg, 2013.
- [128] P. Maymounkov and D. Mazières. Kademia: A Peer-to-Peer Information System Based on the XOR Metric. pages 53–65, 2002.
- [129] J. McKendrick. Marketing chiefs not prepared for data explosion: IBM study, October 2011. [Online viewed on January 24, 2013]

<http://www.smartplanet.com/blog/business-brains/marketing-chiefs-not-prepared-for-data-explosion-ibm-study/19518>.

- [130] L. Mearian. Data growth remains IT's biggest challenge, Gartner says, November 2010. [Online viewed on January 24, 2013]
http://www.computerworld.com/s/article/9194283/Data_growth_remains_IT_s_biggest_challenge_Gartner_says.
- [131] B. Mehta, S. Nangia, M. Gupta, and W. Nejdl. Detecting image spam using visual features and near duplicate detection. In *Proceedings of the 17th International Conference on World Wide Web, (WWW '08), Beijing, China, 21-25 April, 2008*, WWW '08, pages 497–506, 2008.
- [132] M. Minsky and S. Papert. *Perceptrons: An Introduction to Computational Geometry*. MIT Press, Cambridge, MA, USA, 1969.
- [133] S. Misra, R. Narayanan, D. Honbo, and A. Choudhary. High-performance distributed data mining. In H. Kargupta, J. Han, P. S. Yu, R. Motwani, and V. Kumar, editors, *Next Generation of Data Mining*, pages 151–170. Chapman & Hall/CRC, 2009.
- [134] T. M. Mitchell. *Machine Learning*. McGraw-Hill Series in Computer Science. WCB/McGraw-Hill, Boston, MA, 1997.
- [135] A. H. Mohammad and R. A. Zitar. Application of Genetic Optimized Artificial Immune System and Neural Networks in Spam Detection. *Applied Soft Computing*, 11(4):3827–3845, Jun 2011.
- [136] A. Montresor and M. Jelasity. PeerSim: A scalable P2P simulator. In *Proceedings of the 9th International Conference on Peer-to-Peer, (P2P'09), Seattle, Washington, USA, 9-11 September 2009*, pages 99–100, Seattle, WA, sep 2009.

-
- [137] B. Moore. ART 1 and Pattern Clustering. In *Proceedings of the 1988 Connectionist Models Summer School, Carnegie Mellon University, 17-26 June, 1988*, pages 174–185, San Mateo, CA, 1988.
- [138] A. H. Muhamad Amin and A. I. Khan. Single-Cycle Image Recognition Using an Adaptive Granularity Associative Memory Network. In W. Wobcke and M. Zhang, editors, *AI 2008: Advances in Artificial Intelligence*, volume 5360 of *Lecture Notes in Computer Science*, pages 386–392. Springer Berlin Heidelberg, 2008.
- [139] D. N. P. Murthy, R. Jiang, and M. Xie. *Weibull models*. Wiley Series in Probability and Statistics. Wiley, Hoboken, New Jersey, 2003.
- [140] B. Nasution and A. I. Khan. A Hierarchical Graph Neuron Scheme for Real-Time Pattern Recognition. *IEEE Transactions on Neural Networks*, 19(2): 212–229, 2008.
- [141] J. Newsome, E. Shi, D. Song, and A. Perrig. The Sybil Attack in Sensor Networks: Analysis & Defenses. In *Proceedings of the 3rd International Symposium on Information Processing in Sensor Networks, (IPSN '04), Berkeley, CA, USA, 26-27 April, 2004*, pages 259–268, 2004.
- [142] L. T. Nguyen, W. G. Yee, D. Jia, and O. Frieder. A Tool for Information Retrieval Research in Peer-to-Peer File Sharing Systems. In *IEEE 23rd International Conference on Data Engineering, (ICDE 2007), Atlanta, USA, 3-7 April, 2006*, pages 1525–1526, April 2007.
- [143] M. M. Oo, T. T. Soe, and A. Thida. Fault tolerance by replication of distributed database in P2P system using agent approach. *International Journal of Computers*, 4:9–18, 2010.
- [144] L. Ozgur, T. Gungor, and F. Gurgen. Spam Mail Detection Using Artificial Neural Network and Bayesian Filter. In Z. Yang, H. Yin, and R. Everson, editors, *Intelligent Data Engineering and Automated Learning IDEAL 2004*,

- volume 3177 of *Lecture Notes in Computer Science*, pages 505–510. Springer Berlin Heidelberg, 2004.
- [145] O. Papapetrou, W. Siberski, and S. Siersdorfer. Collaborative classification over P2P networks. In *Proceedings of the 20th International Conference Companion on World Wide Web, (WWW '11), Hyderabad, India, 28 March - 1 April, 2011*, WWW '11, pages 97–98, 2011.
- [146] P. Parvathipuram, V. Kumar, and G.-C. Yang. An efficient leader election algorithm for mobile ad hoc networks. In *Proceedings of the First international conference on Distributed Computing and Internet Technology, (ICDCIT'04), Bhubaneswar, India, 22-24 Dec, 2004*, pages 32–41, 2004.
- [147] S. S. Patil and H. S. Patil. Study and Review of Various Image Texture Classification Methods. *International Journal of Computer Applications*, 75 (16):33–38, August 2013.
- [148] M. J. Pazzani. A Framework for Collaborative, Content-Based and Demographic Filtering. *Artificial Intelligent Review*, 13(5-6):393–408, Dec. 1999.
- [149] G. Pitsilis, P. Periorellis, and L. Marshall. A policy for electing super-nodes in unstructured P2P Networks. In *Proceedings of the Third International Conference on Agents and Peer-to-Peer Computing, (AP2PC'04), New York, NY, USA, 19 July, 2004*, AP2PC'04, pages 54–61, 2004.
- [150] M. J. D. Powell. *The Theory of Radial Basis Function Approximation in 1990*, pages 105–210. Oxford University Press, USA, May 1992.
- [151] H. Qin and S. Tang. A Solution to Dimensionality Curse of BP Network in Pattern Recognition Based on RS Theory. In *Proceedings of the 2009 International Joint Conference on Computational Sciences and Optimization, (CSO '09), Hainan Island, China, 24 - 26 April, 2009*, CSO '09, pages 636–638, 2009.

-
- [152] D. Qiu and R. Srikant. Modeling and Performance Analysis of BitTorrent-like Peer-to-Peer Networks. In *Proceedings of the 2004 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications, (SIGCOMM '04), Portland, Oregon, USA, 20 August - 3 September, 2004*, pages 367–378, 2004.
- [153] J. R. Quinlan. Bagging, boosting, and C4.5. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence, (AAAI 1996), Portland, Oregon, 4-8 August, 1996*, pages 725–730, August 1996.
- [154] S. Ratnasamy, P. Francis, M. Handley, R. M. Karp, and S. J. Shenker. A Scalable Content-addressable Network. *Computer Communication Review*, 31:161–172, 2001.
- [155] M. Ripeanu. Peer-to-Peer architecture case study: Gnutella network. In *Proceedings First International Conference on Peer-to-Peer Computing, Linkping, Sweden, 27-29 August 2001*, pages 99–100, 2001.
- [156] R. Rodrigues and B. Liskov. High Availability in DHTs: Erasure Coding vs. Replication. In *Proceedings of the 4th International Conference on Peer-to-Peer Systems, (IPTPS'05), Ithaca, NY, 24-25 February, 2005, IPTPS'05*, pages 226–239, 2005.
- [157] L. Rokach. Ensemble-based classifiers. *Artificial Intelligent Review*, 33(1-2): 1–39, Feb. 2010.
- [158] F. Rosenblatt. The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain. *Psychological Reviews*, 65(6):386–408, November 1958.
- [159] M. Rossi. java.sizeOf, July 2014. [Online posted on July 8, 2014] <http://sizeof.sourceforge.net>.

-
- [160] A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale Peer-to-Peer systems. In *2001 In IFIP/ACM International Conference on Distributed Systems Platforms (Middleware), Heidelberg, Germany, 12-16 November, 2001*, 2001.
- [161] G. Ruan and Y. Tan. A three-layer back-propagation neural network for spam detection using artificial immune concentration. 14(2):139–150, 2010.
- [162] S. Rueger. Multimedia information retrieval. In *Proceedings of the 33rd International ACM SIGIR Conference on Research and Development in Information Retrieval, Geneva, (SIGIR '10), Switzerland, 19- 23 July, 2010*, pages 906–906, 2010.
- [163] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. *Learning internal representations by error propagation*, pages 318–362. MIT Press, Cambridge, MA, USA, 1986.
- [164] D. Saad. *On-Line Learning in Neural Networks*. Publications of the Newton Institute. Cambridge University Press, 2009.
- [165] S. K. Saha, A. K. Das, and B. Chanda. Image retrieval based on indexing and relevance feedback. *Pattern Recognition Letters*, 28(3):357–366, Feb. 2007.
- [166] M. Sahami, S. Dumais, D. Heckerman, and E. Horvitz. A Bayesian Approach to Filtering Junk E-Mail. In *1998 AAAI Workshop on Learning for Text Categorization, Madison, Wisconsin, 27 July, 1998*, July 1998.
- [167] C. Sanden and J. Z. Zhang. Enhancing multi-label music genre classification through ensemble techniques. In *Proceedings of the 34th International ACM SIGIR Conference on Research and Development in Information Retrieval, (SIGIR '11), Beijing, China, 25-29 July, 2011*, pages 705–714, 2011.

-
- [168] S. Saroiu, K. Gummadi, R. Dunn, S. Gribble, and H. Levy. An analysis of Internet content delivery systems. In *Proceedings of 5th Symposium on Operating Systems Design and Implementation, (OSDI'02), Boston, Massachusetts, USA, 911 December, 2002*, December 2002.
- [169] R. E. Schapire, Y. Freund, P. Bartlett, and W. S. Lee. Boosting the Margin: A New Explanation for the Effectiveness of Voting Methods. *The Annals of Statistics*, 26(5):1651–1686, 1998.
- [170] B. Schölkopf and A. Smola. *Learning with kernels: support vector machines, regularization, optimization, and beyond*. Adaptive computation and machine learning. MIT Press, 2002.
- [171] J. Seedorf. Security Issues for P2P-based Voice-and Video-streaming Applications. *iNetSec 2009 Open Research Problems in Network Security*, pages 95–110, 2009.
- [172] J. S. Seo and S. Lee. Fast communication: Higher-order moments for musical genre classification. *Signal Processing*, 91(8):2154–2157, Aug. 2011.
- [173] C. Shan. Learning local binary patterns for gender classification on real-world face images. *Pattern Recognition Letters*, 33(4):431–437, March 2012.
- [174] F. Shirazi. Free and Open Source Software versus Internet content filtering and censorship: A case study. *Journal of System Software*, 85(4):920–931, Apr. 2012.
- [175] X. Shu and X. Li. Tambour: A scalable and robust DHT protocol. In *Proceedings 1st International Conference on Information Science and Engineering (ICISE), Nanjing, China, 26-28 December 2009*, pages 417–420, Dec 2009.
- [176] Sig2Dat. Sig2dat website, April 2012. [Online viewed on April 6, 2012] <http://sourceforge.net/projects/sig2dat/>.

-
- [177] Skype. Skype website, April 2011. [Online viewed on April 6, 2012]
<http://www.skype.com/intl/en/home>.
- [178] Sopcast. Sopcast website, April 2011. [Online viewed April 6, 2012]
<http://www.sopcast.org/>.
- [179] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable Peer-to-Peer lookup service for Internet applications. *SIGCOMM Computing Communication Review*, 31:149–160, August 2001.
- [180] N. Stojnic, L. Probst, and H. Schuldt. COMPASS - Optimized Routing for Efficient Data Access in Mobile Chord-based P2P Systems. In *Proceedings of 14th International Conference on Mobile Data Management, Milan, Italy, 3-6 June, 2013*, pages 46–55, 2013.
- [181] D. Stutzbach and R. Rejaie. Understanding churn in Peer-to-Peer networks. In *Proceedings of the 6th ACM SIGCOMM on Internet measurement, Rio de Janeiro, Brazil, 25 - 27 October, 2006*, pages 189–202, October 2006.
- [182] M. Sugisaka. Fast pattern recognition using a neurocomputer. *Artificial Life and Robotics*, 1:69–72, 1997.
- [183] N. Sundararajan and P. Saratchandran. *Parallel architectures for artificial neural networks: paradigms and implementations*. Systems Series. IEEE Computer Society, Los Alamitos, CA, USA, 1st edition, 1998.
- [184] R. W. Thommes and M. Coates. Epidemiological Modelling of Peer-to-Peer Viruses and Pollution. In *Proceedings of 25th IEEE International Conference on Computer Communications, (INFOCOM 2006), Barcelona, Spain, 23 -29 April, 2009*, pages 1–12, April 2006.
- [185] S. T. Toborg and K. Hwang. Cooperative vision integration through data-parallel neural computations. *IEEE Transaction Computers*, 40:1368–1379, December 1991.

-
- [186] L. Toka, M. Dell’Amico, and P. Michiardi. Data transfer scheduling for p2p storage. In *2011 IEEE International Conference on Peer-to-Peer Computing (P2P), Tokyo, Japan, 31 August- 2 September, 2011*, pages 132–141, August 2011.
- [187] J. Torresen, S. Mori, H. Nakashima, S. Tomita, and O. Landsverk. Exploiting parallel computers to reduce neural network training time of real applications. In C. Polychronopoulos, K. Joe, K. Araki, and M. Amamiya, editors, *High Performance Computing*, volume 1336 of *Lecture Notes in Computer Science*, pages 405–414. Springer Berlin / Heidelberg, 1997.
- [188] C.-C. Tseng, J.-C. Chen, C.-H. Fang, and J.-J. James Lien. Human action recognition based on graph-embedded spatio-temporal subspace. *Pattern Recognition*, 45(10):3611 – 3624, 2012.
- [189] G. Tsoumakas and I. Katakis. Multi-label classification: An overview. *International Journal of Data Warehousing and Mining*, 2007:1–13, 2007.
- [190] M. Tuceryan and A. K. Jain. Texture analysis. In C. H. Chen, L. F. Pau, and P. S. P. Wang, editors, *Handbook of Pattern Recognition and Computer Vision*, pages 235–276. World Scientific Publishing Co., Inc., River Edge, NJ, USA, 1993.
- [191] Tudzu. Tudzu website, April 2012. [Online viewed on April 6, 2012]
<http://www.tudzu.com/en/>.
- [192] TVUPlayer. Tvu networks website, April 2012. [Online viewed April 6, 2012].
- [193] G. Tzanetakis and P. Cook. Musical genre classification of audio signals. *IEEE Transactions on Speech and Audio Processing*, 10(5):293–302, 2002.
- [194] M. Ursino, E. Magosso, and C. Cuppini. Recognition of Abstract Objects Via Neural Oscillators: Interaction Among Topological Organization, Associative Memory and Gamma Band Synchronization. *IEEE Transactions on Neural Networks*, 20(2):316 –335, feb. 2009.

-
- [195] R. Van Renesse, K. P. Birman, and W. Vogels. Astrolabe: A robust and scalable technology for distributed system monitoring, management, and data mining. *ACM Transactions on Computer Systems*, 21(2):164–206, 2003.
- [196] A. B. Vieira, S. Campos, and J. Almeida. Fighting attacks in P2P live streaming: simpler is better. In *Proceedings of the 28th IEEE International Conference on Computer Communications Workshops, (INFOCOM'09), Rio de Janeiro, Brazil, 19-25 April, 2009*, pages 355–356, 2009.
- [197] K. Walsh and E. G. Sirer. Thwarting P2P pollution using object reputation. Technical Report cul.cis/TR2005-1980, Cornell University, 2005.
- [198] K. Walsh and E. G. Sirer. Experience with an object reputation system for Peer-to-Peer filesharing. In *Proceedings of the 3rd conference on Networked Systems Design & Implementation, (NSDI'06), Boston, MA, 22-24 April, 2009*, pages 1 – 1, 2006.
- [199] A. Wang. The Shazam music recognition service. *Communications ACM*, 49(8):44–48, Aug. 2006.
- [200] D.-H. Wang, C.-L. Liu, and X.-D. Zhou. An approach for real-time recognition of online Chinese handwritten sentences. *Pattern Recognition*, 45(10):3661 – 3675, 2012.
- [201] F. Wang, Y. Mo, and B. Huang. P2p-avs: P2p based cooperative voip spam filtering. In *Proceedings of IEEE Wireless Communications and Networking Conference, (WCNC 2007), Hong Kong, China, 11-15 March, 2007*, pages 3547–3552, March 2007.
- [202] F. Wang, F. Wang, B. Huang, and L. Yang. ADVS: a reputation-based model on filtering SPIT over P2P-VoIP networks. *The Journal of Supercomputing*, 64(3):744–761, 2013.
- [203] H. Weatherspoon and J. Kubiatowicz. Erasure Coding Vs. Replication: A Quantitative Comparison. In *Revised Papers from the First International*

-
- Workshop on Peer-to-Peer Systems (IPTPS '01), Cambridge, MA, USA, 7-8 Mar, 2002*, pages 328–338, 2002.
- [204] P. Wei, Q. Hu, P. Ma, and X. Su. Robust feature selection based on regularized brownboost loss. *Knowledge Based System*, 54:180–198, Dec. 2013.
- [205] Windows. Windows messenger website, April 2013. [Online viewed on April 7, 2013]
<http://explore.live.com/messenger>.
- [206] S. Wolf and P. Merz. Evolutionary local search for the super-peer selection problem and the p-hub median problem. In T. Bartz-Beielstein, M. Blesa Aguilera, C. Blum, B. Naujoks, A. Roli, G. Rudolph, and M. Sampels, editors, *Hybrid Metaheuristics*, volume 4771 of *Lecture Notes in Computer Science*, pages 1–15. Springer Berlin Heidelberg, 2007.
- [207] R. Wolff and A. Schuster. Association Rule Mining in Peer-to-Peer Systems. 34(6):2426–38, Dec 2004.
- [208] R. Wolff, K. Bhaduri, and H. Kargupta. Local L2-Thresholding Based Data Mining in Peer-to-Peer Systems. In *Proceedings of the Sixth SIAM International Conference on Data Mining, Bethesda, Maryland, USA, April 20-22, 2006*, pages 430–441, April 2006.
- [209] D. H. Wolpert. Stacked generalization. *Neural Networks*, 5:241–259, 1992.
- [210] Wuala. Wuala website, April 2013. [Online viewed on April 9, 2013]
<http://www.wuala.com/>.
- [211] Y. Xia, S. Chen, and V. Korgaonkar. Load Balancing with Multiple Hash Functions in Peer-to-Peer Networks. In *Proceedings of the 12th International Conference on Parallel and Distributed Systems, (ICPADS '06), Minneapolis, Minnesota, USA, 12-15 July 2006*, pages 411–420, 2006.

-
- [212] B. Xiao, J.-F. Ma, and J.-T. Cui. Combined blur, translation, scale and rotation invariant image recognition by Radon and pseudo-Fourier-Mellin transforms. *Pattern Recognitions*, 45(1):314–321, Jan 2012.
- [213] Yahoo. Yahoo messenger website, April 2013. [Online viewed on April 6, 2013].
- [214] K.-P. Yoon and C.-L. Hwang. *Multiple Attribute Decision Making: An Introduction (Quantitative Applications in the Social Sciences)*. SAGE Publications, Inc, California, January 1995.
- [215] Z.-J. Zha, L. Yang, T. Mei, M. Wang, Z. Wang, T.-S. Chua, and X.-S. Hua. Visual query suggestion: Towards capturing user intent in internet image search. *ACM Transaction Multimedia Computing Communication Application*, 6(3):13:1–13:19, Aug. 2010.
- [216] H. Zhang, C. X. Ling, and Z. Zhao. The learnability of naive bayes. In *Proceedings of Canadian Artificial Intelligence Conference, Victoria, Canada, 9-11 May, 2005*, pages 432–441, 2005.
- [217] H. Zhang, B. Zhang, W. Huang, and Q. Tian. Gabor wavelet associative memory for face recognition. *IEEE Transactions on Neural Networks*, 16(1):275–278, January 2005.
- [218] B. Y. Zhao, J. D. Kubiawicz, and A. D. Joseph. Tapestry: An Infrastructure for Fault-tolerant Wide-area Location and Routing. Technical Report UCB/CSD-01-1141, EECS Department, University of California, Berkeley, Apr 2001.
- [219] P. Zheng, J. Zhang, and W. Tang. Learning associative memories by error backpropagation. *IEEE Transactions on Neural Networks*, 22(3):347–355, march 2011.
- [220] F. Zhou, L. Zhuang, B. Y. Zhao, L. Huang, A. D. Joseph, and J. Kubiawicz. Approximate Object Location and Spam Filtering on Peer-to-peer

- Systems. In *Proceedings of the ACM/IFIP/USENIX 2003 International Conference on Middleware, (Middleware '03), Rio de Janeiro, Brazil, 16-20 June, 2003*, pages 1–20, 2003.
- [221] R. Zhou and K. Hwang. Gossip-based Reputation Aggregation for Unstructured Peer-to-Peer Networks. *International Parallel and Distributed Processing Symposium, (IPDPS 2007), Long Beach, California, USA, 26 - 30 March, 2007*, pages 1–10, 2007.
- [222] M. Zuo, Y.-H. Ma, R. Chbeir, and J.-H. Li. Combating P2P File Pollution with Co-alerting. In *Proceedings of the 2007 Third International IEEE Conference on Signal-Image Technologies and Internet-Based System, (SITIS 2007), Shanghai, China, 16-19 December, 2007*, pages 289–297, 2007.