CORA Cork Open Research Archive
Cartlann Taighde Oscailte Chorcaí

Title	Asynchronous distributed clustering algorithms for wireless mesh network
Authors	Qiao, Cheng
Publication date	2021-04-10
Original Citation	Qiao, C. 2021. Asynchronous distributed clustering algorithms for wireless mesh network. PhD Thesis, University College Cork.
Type of publication	Doctoral thesis
Rights	© 2021, Cheng Qiao. - https://creativecommons.org/licenses/by-nc-nd/4.0/
Download date	2026-11-05 12:27:06
Item downloaded from	https://hdl.handle.net/10468/11300



UCC
University College Cork, Ireland
Coláiste na hOllscoile Corcaigh

Asynchronous distributed clustering algorithms for wireless mesh network

Cheng Qiao
114220132

Thesis submitted for the degree of Doctor of Philosophy



NATIONAL UNIVERSITY OF IRELAND, CORK

College of Science, Engineering and Food Science

School of Computer Science & Information Technology
Insight Centre for Data Analytics

April 2021

Head of Department: Prof. Cormac J. Sreenan

Supervisors: Prof. Kenneth N.Brown,
Prof. Barry O'Sullivan

Research supported by Science Foundation Ireland

Contents

List of Figures iv
List of Tables ix
Acknowledgements xii
Abstract xiii

1 Introduction **1**
1.1 Thesis Statement 3
1.2 Key Contributions 3
1.3 Thesis Organization 4
1.4 List of Publication 5

2 Background and Literature Review **6**
2.1 Clustering 6
2.1.1 Partition Based Clustering 8
2.1.2 Graph Theory Based Clustering 12
2.1.3 Hierarchy Based Clustering 16
2.2 Selecting the number of clusters 18
2.2.1 Elbow method 19
2.2.2 Affinity Propagation 20
2.2.3 Silhouette method 22
2.3 Similarity measurement 24
2.3.1 Distance based similarity 24
2.3.2 Feature based similarity 25
2.3.3 Earth Mover's Distance based similarity 26
2.4 Information Dissemination in Wireless Networks 28
2.4.1 Flooding 28
2.4.2 Gossip and weighted gossip algorithm 29
2.4.3 SPIN 31
2.5 Model selections 33
2.6 Distributed learning frameworks for wireless networks 34
2.6.1 Local learning 36
2.6.2 Information exchange and data aggregation 39
2.6.3 Convergence 44
2.6.4 Weaknesses of the state of the art and corresponding solutions 45

3 Asynchronous Distributed Clustering for Mesh Networks **48**
3.1 Main Framework 49
3.2 Cluster descriptions and integration 49
3.3 Distributed Clustering algorithm and various scenarios 54
3.3.1 Agent knowledge: known number of nodes 55
3.3.2 Agent knowledge: direct neighbourhood only 56
3.3.3 Time Complexity 57
3.4 Empirical Evaluation 59
3.5 Significance Study 63

3.6 Conclusion and future directions 65

4 Adaptive Asynchronous Distributed Clustering for Mesh Networks 67
4.1 Distributed Algorithms 68
4.2 Centralised methods 70
4.3 Time Complexity 71
4.4 Measurement 72
4.5 Experimental Results 73
4.5.1 K-means as clustering algorithm 73
4.5.2 SG as clustering algorithm 79
4.5.3 GMM as clustering algorithm 80
4.5.4 Summary 86
4.6 Significance Study 87
4.7 Conclusion 88

5 Identifying multiple patterns in distributed clustering 91
5.1 Research question and proposed solutions 92
5.2 An EMD based method to measure the similarity between cluster models 93
5.2.1 Standard EMD based method for similarity measurement 98
5.2.2 Optimised EMD based method for similarity measurement 100
5.3 Sharing strategy for different scenarios 108
5.4 Measurement 111
5.4.1 Accuracy against patterns with known assignment (R2) . 111
5.4.2 Accuracy against patterns without known assignment (R1) 112
5.5 Time Complexity 115
5.6 Empirical Evaluation 116
5.6.1 Experiment on well-separated clusters 116
5.6.2 Experiment on overlapped clusters 117
5.6.2.1 K-means as clustering algorithm 121
5.6.2.2 GMM as clustering algorithm 126
5.6.2.3 SG as clustering algorithm 126
5.6.3 Summary 133
5.7 Increasing accuracy against pattern when there is a singleton agent 133
5.7.1 Cluster model between different agents are similar 134
5.7.2 Cluster model between different agents are dissimilar . . 141
5.8 Significance Study 143
5.9 Conclusion 144

6 Detecting single or multiple patterns 147
6.1 Research Problem 148
6.2 Cluster-level similarity based methods 149
6.3 Agent-level similarity based method 150
6.3.1 Kernel Density Estimation 152
6.3.2 Information criterion 155
6.4 Comparison of different methods. 156

6.4.1 K-means as clustering algorithm . . . 156
6.4.2 GMM as clustering algorithm . . . 157
6.4.3 SG as clustering algorithm . . . 159
6.4.4 Summary . . . 161
6.5 Accuracy against data point with different predictions . . . 161
6.6 Conclusion . . . 163

7 Conclusions and Future Work 164
7.0.1 Conclusions . . . 164
7.0.2 Future work . . . 166

8 Appendices 168
8.1 Evaluation of information exchange method . . . 169
8.1.1 Measurement with Dataset Generated on Standard Basis 169
8.1.2 Measurement with Different Number of Data Points . . . 171
8.1.3 Measurement with Different Underlying Distributions . . 172
8.1.4 Measurement with Different Percentiles . . . 174
8.1.5 Possible issue caused by regeneration . . . 175
8.1.6 Summary . . . 178
8.2 Comparison result on single pattern sparse networks . . . 182
8.2.1 K-means as clustering algorithm . . . 182
8.2.2 SG as clustering algorithm . . . 185
8.2.3 GMM as clustering algorithm . . . 188
8.3 Comparison result on similarity measurements . . . 191
8.3.1 GMM as clustering algorithm . . . 191
8.3.2 SG as clustering algorithm . . . 193
8.4 Predict the sub-pattern assignment of agent . . . 197
8.5 Comparison result on multiple sparse networks . . . 199
8.5.1 K-means as clustering algorithm . . . 199
8.5.2 GMM as clustering algorithm . . . 200
8.5.3 SG as clustering algorithm . . . 200

List of Figures

1.1 Architecture of wireless mesh network [Akyildiz et al., 2005] . . 1
1.2 An example of learning the wider network pattern by message exchanging 2

2.1 centres of sub-areas displayed on a Google map 7
2.2 Traffic flow in each sub-area 7
2.3 Compasion of K-means and GMM when underlying dataset is generated by symmetric Gaussians 11
2.4 Comparison of K-means and GMM when underlying dataset is generated by asymmetric Gaussians 12
2.5 Groups found by spectral clustering 15
2.6 An example of K-means, GMM and Spectral clustering 16
2.7 Dendrogram when ward's method is applied 18
2.8 Clustering with different number of clusters on Iris dataset . . . 19
2.9 Test data and line chart with different number of clusters 20
2.10 Estimated number of clusters obtained by AP method 22
2.11 Number of clusters found by silhouette 23
2.12 Wavelet EMD [Shirdhonkar and Jacobs, 2008] 27
2.13 Robust EMD [Pele and Werman, 2009] 28
2.14 SPIN protocol [Patel and Parekh, 2014] 32
2.15 Distributed learning framework: (a) with a fusion center. (b) with in-network processing [Predd et al., 2006] 35
2.16 An example of obtaining boundary points of a cluster 38
2.17 An example of Cluster-based topology [Gui et al., 2016] 39
2.18 An example of Chain-based topology 40
2.19 An example of tree-based aggregation process 44

3.1 Information propagation 51
3.2 Original data and its summary description 52
3.3 Bounding boxes received and regenerated data based on it . . . 54
3.4 Comparison of MMF1 method in 10-agent dense network, showing mean results for each measure over 50 runs. 62
3.5 Comparison of different schemes under MMF1 method in a 10-agent sparse network, showing mean results for each measure over 50 runs. 63
3.6 MMF2 method in 10-agent dense networks 64
3.7 Comparison of different schemes under MMF2 method in a 10-agent sparse network 64

4.1 Different scenarios considered in this chapter 71
4.2 Fully symmetric Gaussians with heavy overlap 73
4.3 Partially symmetric Gaussians with slight overlap 73
4.4 Well separated uniform datasets 73
4.5 An example of different overlaps between clusters 73

4.6 K-means with symmetric multi-dimensional Gaussians on dense network . . . 75
4.7 K-means on dense network with asymmetric multi-dimensional Gaussians . . . 77
4.8 K-means on dense network with uniform distributed datasets . . 78
4.9 SG on dense network with symmetric multi-dimensional Gaussians 79
4.10 SG on dense network with asymmetric multi-dimensional Gaussians 81
4.11 SG on dense network with uniform distributed dataset 82
4.12 GMM on dense network with symmetric multi-dimensional Gaussians . . . 83
4.13 GMM on dense network with asymmetric multi-dimensional Gaussians . . . 84
4.14 GMM on dense network with uniform distributed dataset 85
4.15 Comparison of different regenerations when underlying dataset is generated by a symmetric Gaussian . . . 86
4.16 Comparison of different regenerations when underlying dataset is generated by a uniformly rectangle distribution . . . 87

5.1 Raw data points for each agent . . . 94
5.2 Centroids for each agent . . . 95
5.3 Hierarchical clustering by different methods . . . 96
5.4 merging process . . . 98
5.5 Inconsistent method . . . 98
5.6 Elbow method . . . 98
5.7 Estimation process by inconsistent and elbow method . . . 98
5.8 Accuracy on data points against ground truth with different similarity metrics . . . 99
5.9 The number of estimated patterns with different similarity metrics 100
5.10 Reversible EMD between agents . . . 101
5.11 Accuracy measures against centralised k-means with symmetric Gaussians . . . 103
5.12 Accuracy against ground truth with symmetric Gaussians 104
5.13 Accuracy against pattern with symmetric Gaussians 104
5.14 Accuracy against centralised k-means with asymmetric Gaussians 105
5.15 Accuracy against ground truth with asymmetric Gaussians . . . 105
5.16 Accuracy against pattern with asymmetric Gaussians 106
5.17 Accuracy against centralised k-means with uniformly distributed dataset . . . 106
5.18 Accuracy against ground truth with uniformly distributed dataset 107
5.19 Accuracy against pattern with uniformly distributed dataset . . 107
5.20 Comparison of convergence time . . . 108
5.21 Different scenarios considered for multiple patterns 110
5.22 Symmetric table built by agents' prediction . . . 113
5.23 Asymmetric table built by agents' prediction . . . 115
5.24 K-means on multiple patterns dense network with fully symmetric well separated clusters . . . 118

5.25 K-means on multiple patterns dense networks with partially symmetric well separated clusters 119
5.26 K-means on multiple patterns dense networks with uniformly well separated clusters 120
5.27 K-means on dense network with multiple patterns and symmetric multi-dimensional Gaussians 123
5.28 K-means on dense network with multiple patterns and asymmetric multi-dimensional Gaussians 124
5.29 K-means on dense network with multiple patterns and uniformly distributed dataset 125
5.30 GMM on dense network with multiple patterns and symmetric multi-dimensional Gaussians 127
5.31 GMM on dense network with multiple patterns and asymmetric multi-dimensional Gaussians 128
5.32 GMM on dense network with multiple patterns and uniformly distributed dataset 129
5.33 SG on dense network with multiple patterns and fully symmetric multi-dimensional Gaussians 130
5.34 SG on dense network with multiple patterns and partially symmetric multi-dimensional Gaussians 131
5.35 SG on dense network with multiple patterns and uniformly distributed dataset 132
5.36 Performance achieved by DBD with K-means as clustering algorithm and asymmetric Gaussians as underlying dataset 135
5.37 Performance achieved by DBD with K-means as clustering algorithm and uniformly distributed dataset as as underlying dataset 136
5.38 Performance achieved by DBD with GMM as clustering algorithm and asymmetric Gaussians as underlying dataset 137
5.39 Performance achieved by DBD with GMM as clustering algorithm and uniformly distributed datasets as underlying dataset 138
5.40 Performance achieved by DBD with SG as clustering algorithm and asymmetric Gaussians as underlying dataset 139
5.41 Performance achieved by DBD with SG as clustering algorithm and uniformly distributed datasets as underlying dataset 140
5.42 Dataset before and after rotation (left to right) 141
5.43 Accuracy against patterns with different similarities between agents 142
5.44 Accuracy against patterns with different similarities between agents 143

6.1 Flow chart 148
6.2 One example of detecting single and multiple patterns 150
6.3 Application of G-means to agents' clusters 150
6.4 Application of DBSCAN to agents' clusters (1) 151
6.5 Application of DBSCAN to agents' clusters (2) 151
6.6 Agent-level similarity for a 5-agent network 152
6.7 Data sequence and its density estimation 153
6.8 Clusters found by KDE 153

6.9 Comparative of various bandwidth estimation 155
6.10 Accuracy achieved with symmetric Gaussians when K-means is used 157
6.11 Accuracy achieved with asymmetric Gaussians when K-means is used 157
6.12 Accuracy achieved with uniformly distributed dataset when K-means is used 158
6.13 Accuracy achieved with symmetric Gaussians when GMM is used 158
6.14 Accuracy achieved with asymmetric Gaussians when GMM is used 159
6.15 Accuracy achieved with uniformly distributed dataset when GMM is used 159
6.16 Accuracy achieved with symmetric Gaussians when SG is used . 160
6.17 Accuracy achieved with asymmetric Gaussians when SG is used 160
6.18 Accuracy achieved with uniformly distributed dataset when SG is used 160
6.19 Accuracy against data points achieved by Optimised KDE when ground truth pattern is single 162
6.20 Accuracy against data points achieved by DBSCAN when ground truth pattern is single 162

8.1 Data generated by normal and uniform in same range 172
8.2 Data re-generated by M1 and M2 (Normal) 173
8.3 Data re-generated by M1 and M2 (uniform) 173
8.4 Dataset used in the experiment 177
8.5 K-means with fully symmetric Gaussians on sparse network . . 182
8.6 K-means on sparse network with partially symmetric Gaussians 183
8.7 K-means on sparse network with uniform distributed dataset . . 184
8.8 SG on sparse network with fully symmetric Gaussians 185
8.9 SG on sparse network with partially symmetric Gaussians . . . 186
8.10 SG on sparse network with uniform distributed dataset 187
8.11 GMM on sparse network with fully symmetric Gaussians 188
8.12 GMM on sparse network with partially symmetric Gaussians . . 189
8.13 GMM on sparse network with uniform distributed dataset . . . 190
8.14 Accuracy against centralised GMM with fully symmetric Gaussians 191
8.15 Accuracy against ground truth with fully symmetric Gaussians . 192
8.16 Accuracy against pattern with fully symmetric Gaussians 192
8.17 Accuracy against centralised GMM with datasets generated from uniform distributions 193
8.18 Accuracy against ground truth with uniformly distributed dataset 193
8.19 Accuracy against pattern with uniformly distributed dataset . . 194
8.20 Accuracy against centralised SG with fully symmetric Gaussians 194
8.21 Accuracy against ground truth with fully symmetric Gaussians . 195
8.22 Accuracy against pattern with fully symmetric Gaussians 195
8.25 Accuracy against pattern with uniformly distributed dataset . . 195
8.23 Accuracy against centralised SG with uniformly distributed dataset 196
8.24 Accuracy against ground truth with uniformly distributed dataset 196

8.26 Performance on datasets generated from fully symmetric Gaussians with K-means as clustering algorithm (1) 197
8.27 Performance on datasets generated from fully symmetric Gaussians with K-means as clustering algorithm (2) 198
8.28 Performance on datasets generated from fully symmetric Gaussians with K-means as clustering algorithm (3) 199
8.29 Performance on datasets generated from fully symmetric Gaussians with K-means as clustering algorithm (4) 199
8.30 K-means on sparse network with multiple patterns and fully symmetric multi-dimensional Gaussians 201
8.31 K-means on sparse network with multiple patterns and partially symmetric multi-dimensional Gaussians 202
8.32 K-means on sparse network with multiple patterns and uniformly distributed dataset 203
8.33 GMM on sparse network with multiple patterns and fully symmetric multi-dimensional Gaussians 204
8.34 GMM on sparse network with multiple patterns and partially symmetric multi-dimensional Gaussians 205
8.35 GMM on sparse network with multiple patterns and uniformly distributed dataset 206
8.36 SG on sparse network with multiple patterns and fully symmetric multi-dimensional Gaussians 207
8.37 SG on sparse network with multiple patterns and partially symmetric multi-dimensional Gaussians 208
8.38 SG on sparse network with multiple patterns and uniformly distributed dataset 209

List of Tables

2.1 Pairwise distance between 10 vertices 14
2.2 Distance matrix sorted by row and column 14
2.3 Similarity matrix obtained by Gaussian Kernel 14

3.1 Notation 61
3.2 Pairwise Tukey test for proposed methods and the state-of-the-art. $\diamond$ denotes the difference of mean. 65

4.1 Notation of different methods and measurements 73
4.2 Test for normality with Shapiro-Wilk's test, only the result with K-means as clustering algorithm is given 88
4.3 Pairwise Tukey test for distributed methods and centralized methods, only the results on dense graph is compared(1). $\diamond$ denotes the difference of mean 89

5.1 Description of different methods 101
5.2 Accuracy measures against centralised k-means 103
5.3 One sample t-test for learned number of patterns, only the results on dense graph with K-means as clustering algorithm is compared 144
5.4 Tukey test for accuracy against patterns with same $\boldsymbol{k}$ and different $\boldsymbol{k}$. Only the results on dense graph with K-means as clustering algorithm is reported. $\diamond$ denotes the difference of mean 145

8.1 Measurement for fixed distributions with standard basis 170
8.2 Measurement for different distributions with standard basis . . 171
8.3 Measures varied with number of data points 171
8.4 Measures varied with underlying distributions 174
8.5 Measures varied with percentiles 175
8.6 Default initial centroids, full K-means initialisation, No scheme to make up the shrink 178
8.7 Default initial centroids, full K-means initialisation, make up the shrink of 100% bounding box only 179
8.8 Default initial centroids, full K-means initialisation, extend the bounding line to half way to its closet point 180
8.9 Default initial centroids, full K-means initialisation, combination of scheme 2 and 3 181

I, Cheng Qiao, certify that this thesis is my own work and has not been submitted for another degree at University College Cork or elsewhere.



Cheng Qiao

To my family

Acknowledgements

All things are difficult before they are easy.

— Thomas Fuller (1710 - 1790)

First of all, I would like to thank my supervisors, Prof. Kenneth N. Brown and Prof. Barry O'Sullivan, for guiding me in pursuing my Ph.D degree. During the course, we had numerous discussions which are full of inspirations and encouragements. This work has not been possible without their consistent support and guidance. In the past 5 years, I learned a lot from them, especially on how to become an independent researcher.

I am grateful to Prof. Yong Zhang, for his suggestions and motivating questions. I worked with him before my Ph.D and it was him who showed me how fun is it to do research and guided me to this path.

Many thanks to my family for their endless love and support. My lovely wife Man, who have been comforting me whenever I was frustrated and accompanying me whenever I was feeling lost. My little princess Sunni who always cheered me up. I would also like to thank my only brother Jun and sister in-law Yumei, for taking good care of our parents during my whole Ph.d period.

Special thanks to another two Chinese Ph.D students Hong Huang and Weipeng Huang in our Lab. Weipeng helped me work through technical challenges and discussing with him alway inspire me a lot. But he went to another University in the second year. That makes me always miss the time we had coffee together in the fourth floor of Western Gateway Building. Hong is the partner to play online game and squash with, and that gives me good relaxation apart from doing research.

I would like to thank my life-long friends Deric Duan, Gary Guo, Daniel Kuang, Zi Wang and Jianwen Kuang for their encouragement and support, and making my time during my studies so wonderful. Gathering together in Shenzhen was one of my most precious memories.

I would like to thank all my colleagues in Insight Centre, especially Dr. Mustafa AI-Bado, Dr. Vincent Armant, Dr. Saim Ghafoor, Mohammed Ibrahim and Saeid Rezaei for their motivation and questions. I am also thankful to Science Foundation of Ireland for supporting my studies through Insight, Insight-2 and Enable (SFI/12/RC/2289-P2 and 16/SP/3804), which are co-funded under the European Regional Development Fund.

Abstract

Wireless Mesh Networks are becoming increasingly important in many applications. In many cases, data is acquired by devices that are distributed in space, but effective actions require a global view of that data across all devices. Transmitting all data to the centre allows strong data analytics algorithms to be applied, but consumes battery power for the nodes, and may cause data overload. To avoid this, distributed methods try to learn within the network, allowing each agent to learn a global picture and take appropriate actions.

For distributed clustering in particular, existing methods share simple cluster descriptions until the network stabilises. The approaches tend to require either synchronous behaviour or many cycles, and omit important information about the clusters. In this thesis, we develop asynchronous methods that share richer cluster models, and we show that they are more effective in learning the global data patterns.

Our underlying method describes the shape and density of each cluster, as well as its centroid and size. We combine cluster models by re-sampling from received models, and then re-clustering the new data sets. We then extend the approach, to allowing clustering methods that do not require the final number of clusters as input. After that, we consider the cases that there might be sub-groups of agents that are receiving different patterns of data. Finally, we extend our approaches to scenarios where each agent has no idea about whether there is a single pattern or are multiple patterns.

We demonstrate that the approaches can regenerate clusters that are similar to the distributions that were used to create the test data. When the number of clusters are not available, the learned number of clusters is close to the ground truth. The proposed algorithms can learn how data points are clustered even when there are multiple patterns in the network. When the number of patterns (single or multiple) is not known in advance, the proposed methods Optimised KDE and DBSCAN preform well in detecting multiple patterns. Although they perform worse in detecting the single pattern, they can still learn how data points are clustered.

Chapter 1

Introduction

A wireless mesh network (WMN) is a communications network made up of radio nodes organised in a mesh topology, communicating data over a multi-hop wireless network, with a wide range of applications [Harb et al., 2017, Ting et al., 2018, Alanazi et al., 2016]. Nodes in the network operate as host (sending packets out) and router (forwarding packets on behalf of other nodes). Figure 1.1 shows the architecture of WMN. In WMN, the network is established with the help of various numbers of wireless nodes to communicate with each other. It also can be connected with various existing networks, including mobile ad hoc network and sensor network, to provide efficient communication.

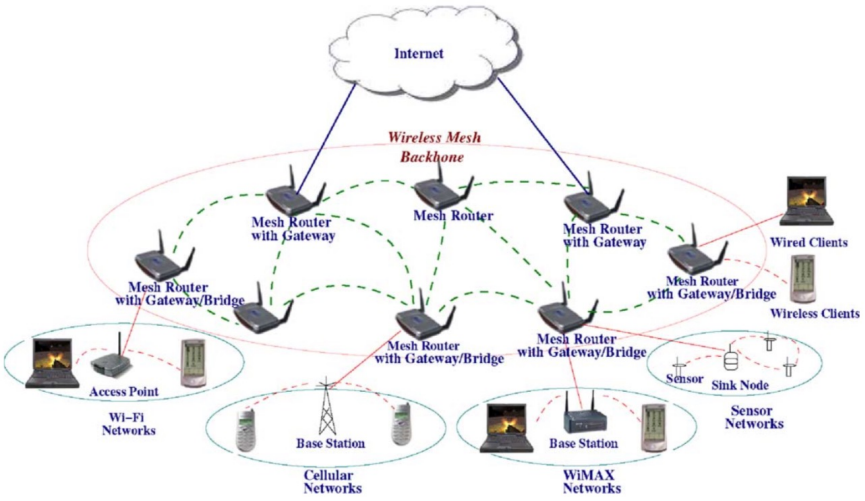


Figure 1.1: Architecture of wireless mesh network [Akyildiz et al., 2005]

As the Internet of Things and Cyber Physical Systems develop, these network nodes must also include actuators, which respond to sensed data or communicated instructions to effect changes to the environment. In some applications, the

actions taken by a node will depend on wider patterns observed in the network - e.g. temperature readings in indoor heating systems, or vehicle movements in traffic control - and so communication must flow back and forwards across the network.

Although a massive amount of data, generated by device such as smart phones and sensors has facilitated the machine learning methods, the shortage of data resource is great due to the following two factors: 1) data are usually isolated. It is disallowed to centralize the data to a central server (owned by different entities) or cloud, or the cost to centralize them is high, 2)the increasing concerns of data privacy and security. Transmitting all data to a central server creates risks for the privacy and security of the data.

To address these concerns, we introduce an algorithm to cluster distributed datasets without centralizing raw data. Each node acts as its own decision maker, but must communicate with other nodes to learn wider network patterns [Zeng et al., 2012]. The message shared among neighbours is a representation of the raw data points. Nodes can infer the original raw data based on the message received. The methods trade off potentially lower decision quality for reduced communication, reduced energy use and improved privacy. For example, there are some sensors deployed in an area in figure 3.1, and each sensor has a different reading v_i $(1 \leq i \leq 9)$. Sensors could learn the average reading $\bar{v}$ by in-networking learning and message exchanging without gathering all data to a central server.

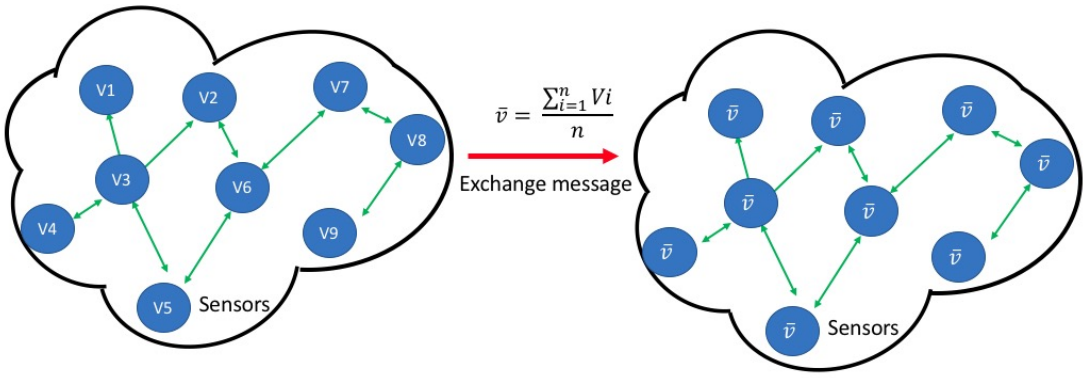


Figure 1.2: An example of learning the wider network pattern by message exchanging

In this work, we focus on clustering, where we attempt to identify patterns in the data being received by the nodes in the network. The intention is that each node should finish with cluster models, describing the data received across the network, while minimising inference time and communication cost, and respecting

the privacy of the raw data. We also want to avoid sharing raw data, but also to respect what may be the privacy of the individual data points. We develop an asynchronous mechanism for sharing information, to avoid synchronisation delays.

We develop different approaches to in-network clustering, examining the trade-offs between sharing local clusters and conducting incremental clustering as information flows through the network. We design and implement different methods for describing the shapes of locally identified clusters, to improve the global clustering performance. We evaluate the methods empirically, using an asynchronous message delay simulator. The underlying datasets we used for simulation are 2-dimensional spatial datasets, which are generated from different distributions. We consider both the flat network topology and tree-based network topology, and experiments show that flat network topology is better for our proposed clustering algorithms. Compared to tree-based network topology, flat network topology does not introduce single point of failure, but achieve similar clustering accuracy and converges faster.

1.1 Thesis Statement

Distributed clustering in wireless mesh networks is possible by using an asynchronous adaptive clustering algorithm without transmitting raw data, and without incurring high latency or high energy costs, but can achieve high clustering quality and even when extended to scenarios where there are sub-groups of agents are receiving data from different environments

1.2 Key Contributions

To address the issues and challenges for learning the wider network patterns in an unknown network, this thesis makes the following contributions:

1. When all agents sense data from the same source, we designed, implemented and evaluated an asynchronous distributed clustering algorithm that allow each agent to learn the global view but respects the privacy and communication cost. This algorithm works well even without knowing the size of network.

2. We relax the assumption that the number of clusters are fixed in advance and each agent estimates the number of clusters locally and independently.

3. To mitigate the computation cost of Earth Mover Distance (EMD) [1] but fully utilise the shape information of dataset owned by individual agent, we proposed a modified EMD method to measure the similarity of cluster models learned by different agents. Compared to the state of the arts, our proposed adapted EMD method converges fast and performs better in measuring the similarity of cluster models.

4. Based on the adapted EMD method, we proposed a hierarchical clustering algorithm to predict the number of patterns when there are sub-groups of agents are receiving data from different environments. Experiments show that this algorithm is robust and works well in both evenly-split patterns and unequally-split patterns (including the case with singleton agent).

1.3 Thesis Organization

The thesis is organized as follows:

In Chapter 2, the current state of the art solutions for learning the wider network pattern are examined and issues and challenges of those solutions are discussed. In addition, technologies and methodologies that used in the thesis are described.

In Chapter 3, we aim to learning the wider network patterns with known number of clusters in single pattern network. A cluster model, that obtained by different clustering approaches including K-means and Gaussian Mixture Models, is shared with neighbours. In addition, a method is proposed to describe the shape of clusters found by K-means. Each agent infer the original data points by the cluster model received.

In Chapter 4, the assumption that number of clusters are known in advance is relaxed, and how to learn the global view is presented. Initially, all agent estimate the number of clusters locally and independently. Then, agents update the number of clusters whenever receive models from neighbours.

In Chapter 5, a potential method to measure the similarity between cluster models learned by different agents is proposed, and the performance is evaluated over the state of the art solutions. Then we describe an approach to learn the wider network patterns for multiple patterns network.

In Chapter 6, we relax the assumption that the algorithm knows there is a single

[1] it measures the distance between two probability distributions and is widely used in Computer Vision.

pattern or multiple patterns in advance. Two types of methods are proposed to detect the single or multiple patterns.

1.4 List of Publication

The list of papers which have already been published from this thesis are:

1. Qiao, Cheng, and Kenneth N. Brown. "Asynchronous Distributed Clustering Algorithm for Wireless Sensor Networks." Proceedings of the 2019 4th International Conference on Machine Learning Technologies. 2019.

Chapter 2

Background and Literature Review

This chapter introduces the technologies and concepts used in this thesis and gives a survey on related work for the distributed learning problem. Three different types of clustering algorithms are discussed in detail first, then commonly used methods to pick the number of clusters for those clustering algorithms are presented, followed by a description of similarity measurement methods. After that, information dissemination in networks are described. Finally, we review the most relevant approaches to the problem of distributed learning.

2.1 Clustering

One of the first steps in understanding a dataset is identifying patterns in the data. Once we have established the patterns, we can begin to identify items that are qualitatively different from each other, and which may have distinct causes; Alternatively, identifying distinct patterns might help improve future decision making and actuation, by creating different cases to be handled differently.

For example, consider a dataset of Uber trips in New York City[1], which provides a large amount of valuable data such as traffic, transit time and peak pickup localities. The dataset shows around 4.5 million Uber pickups in New York City from April to September 2014 and the date, time, latitude and longitude are recorded for each pickup. The analysis of these GPS data can allow cities to optimise traffic flows based on real-time traffic information. From this dataset,

[1] `https://www.kaggle.com/fivethirtyeight/uber-pickups-in-new-york-city`

we could try to infer which hours of the day and which area had the highest number of pickups. This could be done by first dividing the whole city into several sub-areas based on the longitude and latitude, then count the pickups occurred in each sub-area by date and time. Figure 2.1 shows the eight sub-area centres displayed on a Google map.

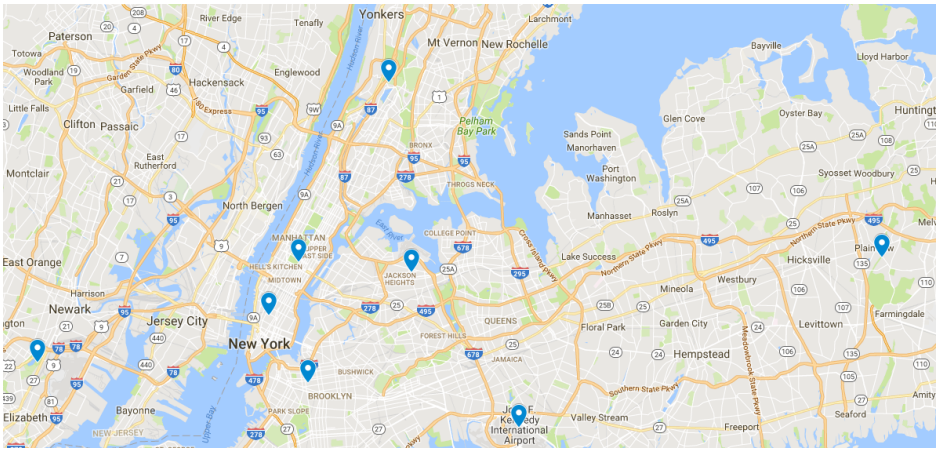


Figure 2.1: centres of sub-areas displayed on a Google map

Then the traffic flow in each sub-area over time is shown in figure 2.2, where the x axis is the hour, the y axis the count, and the colours are the different sub-areas. In this case, location information are used to define the patterns of traffic flows. The pick up localities inside the same sub-area are close to each other while different from pick up locations in other sub-areas.

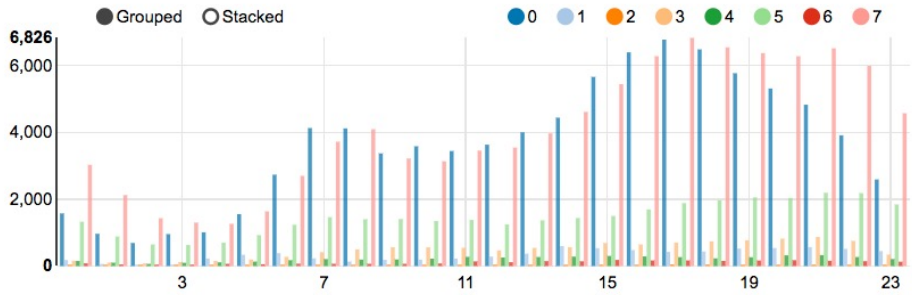


Figure 2.2: Traffic flow in each sub-area

These patterns are called "clusters". Clustering algorithms attempt to identify clusters automatically in the data. Typically, the algorithm is not given any exemplars or descriptions of the clusters, and there is no oracle that can report whether or not an item has been correctly clustered. In machine learning terms, the clustering algorithm is "unsupervised". The general aim is to group instances into clusters such that instances in the same clusters are similar to each other

while being different from instances in other clusters. The definitions of similarity and difference, and the methods for forming the clusters, vary between different clustering algorithms and frameworks.

There is no complete definition for clustering, but a classic one is described as follows [Jain and Dubes, 1988]: a clustering algorithm aims to group the instances into separated clusters such that instances in same cluster must as similar as possible while instances in different clusters must be as different as possible and the measurement for similarity and dissimilarity must be clear and have practical meaning.

Clustering algorithms are frequently used in various data-intensive applications. For instance, a retailer could detect groups of customers with similar buying patterns by tracking their purchases. In computational biology, groups of genes could be detected by exhibition of similar behaviour, which may indicate that they are part of the same biological pathway. Xu et al. [Xu and Tian, 2015] presented a comprehensive survey of various clustering algorithms.

In this section, we discuss three types of clustering algorithm applied in this thesis: partition based, graph theory based and hierarchy based.

2.1.1 Partition Based Clustering

The K-means algorithm is a widely applied clustering technique [Likas et al., 2003]. Given a dataset with n points in a d-dimensional space and the number of clusters $\boldsymbol{k}$ to find, the goal of K-means is to partition the data points into $\boldsymbol{k}$ clusters which best describe the data distribution. The output is a set of representative locations in space, one for each cluster, such that each point is associated with its nearest cluster, and the locations, or centroids, are the weighted average of the locations of its associated points. The error for each original data points is the Euclidean distance to its centroids and the error for the cluster is the sum of squared errors. The objective of K-means is then to minimise the sum of cluster errors:

$$E = \sum_{k=1}^{k} \sum_{p \in C_i} (|p - m_i|)^2 \tag{2.1}$$

where E denotes total squared error, p is a data point and m_i is the centroid cluster C_i.

Algorithm 1 shows the main steps for K-means algorithm. It starts with k initial

centroids, chosen by some method, and a partition is created by assigning each instance to its closest cluster. Then each cluster centroid is updated to be the mean value of all its members. These steps will be repeated until the cluster membership has stabilised.

Algorithm 1: K-means algorithm

```
1 Input: Set of instances: D = {x_0, x_1, ··· , x_n}, number of desired clusters k;
2 Output: Set of clusters C;
3 Choose k initial centroids {c_0, c_1, , ··· , c_k};
4 while cluster membership is not stabilised do
5     Create a new partition by assigning each instance to its closest centroid;
6     Compute new cluster centroids;
```

K-means is a computationally fast algorithm and easy to implement. It provides a hard assignment of points to clusters, where a point either totally belongs to a cluster or not at all. However, its weaknesses include:

1. the number of clusters $\boldsymbol{k}$ has to be specified in advance.
2. it is sensitive to noisy data and outliers.
3. it assumes clusters can be separated by a convex hull.

K-means can be viewed as creating a Voronoi partition, where each cluster is at the centre of its Voronoi cell. Any point outside this partition is not considered a member of the cluster. Depending on the representative objects and optimisation objective, there are different variants of K-means, including K-median [Kariv and Hakimi, 1979], K-medoids [Kaufman and Rousseeuw, 2009] and K-center [Dyer and Frieze, 1985].

1. K-median: Instead of calculating the mean for each cluster to determine its centroid, the centroid is calculated as the value in the median for each attribute. The objective is to minimise the sum of distance from each instance to its nearest median. The advantage of K-medians is that each representative is constructed from data values for each dimension that actually appear for some points in the cluster.
2. K-medoids: rather than calculate the mean or median of the data points, K-medoids instead selects the data point which minimises the sum of squared distance to all other points in the cluster. The advantage is that the representative is then an actual data point.

3. K-center: Similar to K-medoids, existing data points are chosen to represent the cluster, but now the point is chosen that minimises the maximum distance to other points in the cluster.

Note that medoids and centers, rather than centroids, are all true instances from the dataset [2]. K-means and K-medoids aim to optimise the compactness (sum of squared distance from instance to cluster representation) while K-median minimise the cluster diameter and K-center minimise the cluster radius [Basu et al., 2008]. Compared to K-means, K-medoids is less sensitive to outliers since it is based on the most centrally located object in a cluster. K-median, K-medoids and K-center assume that clusters can be separated by a convex hull and the number of clusters has to be specified in advance.

However, in some cases, we might not have complete confidence in the cluster assignment of points located in the margin area between two clusters. This motivates the introduce of a soft clustering method Gaussian Mixture Model (GMM) clustering: fit data points to a GMM, and then turn it into a hard model by selecting the most likely cluster for each observed points. GMM attempts to find a mixture of multi-dimensional Gaussian probability distributions that best model any input dataset. Each cluster is modelled as a different Gaussian distribution and each data point could have been generated by any of those distribution with a probability. A GMM can be used to create a hard clustering, by turning the probabilities into a membership function, by assigning each point to the Gaussian mean for which it has highest probability.

Suppose there are n observations $\{X_1, X_2, \cdots, X_n\}$ and each X_i is sampled from a mixture model of K mixture components. How could we know which Gaussian generated a particular data point? The GMM is a mixture model that uses a combination of Gaussian probability distributions, so the estimation of the mean and standard deviation parameters for each is required. The mixture component to generate the data point, that is not observable, represents a latent variable. As such, the expectation-maximisation (EM) algorithm is an appropriate approach to use to estimate the parameters of the mixture component. In the EM algorithm, the estimation-step would estimate the latent variable for each data point, and the maximisation step would optimise the parameters of the probability distributions. The process is repeated until a minimum log likelihood is achieved. Algorithm 2 shows the main steps for the GMM.

K-means represents each cluster by its mean, while GMM provides the mean and

[2] Note that medians might be true instances.

Algorithm 2: Gaussian Mixture Model

Input: Set of instances: D = $\{X_0, X_1, \cdots, X_n\}$, K mixture components that generate D;
Output: Set of clusters $\{c_1, c_2, \cdots, c_K\}$;
Randomly assign mean μ_k, covariance ${\delta_k}^2$ and density π_k for each cluster;
while *The objective log likelihood is not minimised* **do**
    /* E step */
    Calculate the probability that it belongs to cluster $c_k (1 \leq k \leq K)$;
        $\gamma_{Z_i}(k) = \frac{P(X_i|Z_i=k)P(Z_i=k)}{P(X_i)} = \frac{\pi_k N(\mu, {\delta_k}^2)}{\sum_{k=1}^{K} N(\mu_k, \delta_k)}$;
    /* M step */
    Compute new mean μ_k for each cluster;
        $\mu_k = \frac{1}{N_k} \sum_{n=1}^{N} \gamma_{Z_i}(k) X_i$;
    Compute new covariance matrix ${\delta_i}^2$ for each cluster ;
        ${\delta_k}^2 = \frac{1}{N_k} \sum_{n=1}^{N} \gamma_{Z_i}(k) X_i (X_i - \mu_k)^2$;
    Compute new density π_k for each cluster ;
        $\pi_k = \frac{N_k}{n}$;

variance, and thus describes the distribution, or shape, of the cluster. However, GMM still suffers from the same problem as K-means that the number of clusters $\boldsymbol{k}$ has to be fixed manually in advance.

Example. Figure 2.3 below shows the different clustering between K-means and GMM. The dataset is randomly generated by four Gaussians with centres $[[-2, 1], [2, 0], [1, 2], [-1, 3]]$. The standard deviation for each Gaussian is fixed to 0.46. There is little difference in terms of centroids found by K-means and mean estimated by GMM (black crosses shown in the figure). However, some points in the margin area between the peach cluster and blue cluster are grouped into the blue cluster by GMM even they are much closer to the centroids of peach cluster. That is because the Gaussians give, for those points, a higher probability of membership.

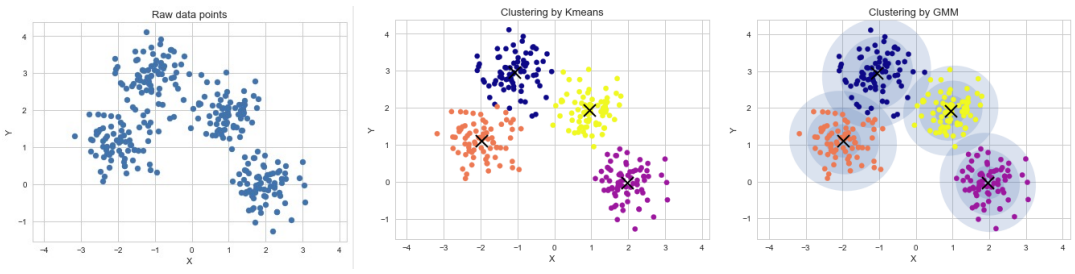


Figure 2.3: Compasion of K-means and GMM when underlying dataset is generated by symmetric Gaussians

Figure 2.4 shows the difference of clusters obtained by K-means and GMM when applied to data generated using a non-circular distribution. It shows that clusters generated by K-means are significantly different from the original distribution, while GMM is generating clusters that are similar to the original distribution.

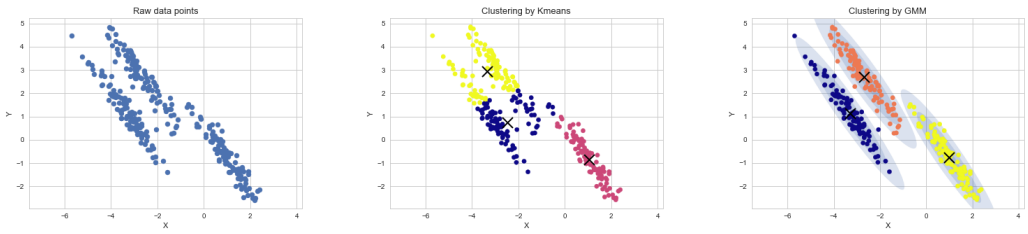


Figure 2.4: Comparison of K-means and GMM when underlying dataset is generated by asymmetric Gaussians

2.1.2 Graph Theory Based Clustering

In last section, each data points x_i is represented by a vector in the space. A similarity graph $G = (V, E)$ is another way to represent data, where the vertices v_i donate the data points. Note that x_i is a vector which denotes the data points while v_i is a vertex without any attributes. Two vertices are connected if the similarity s_{ij} between vertex v_i and v_j is larger than a threshold and the edge is weighted by s_{ij}. Intuitively, the goal of clustering is to divide data points into several groups such that points in the same group are similar and points in different groups are dissimilar. If data is represented by similarity graph, now the clustering problem could be reformulated as: find a partition of the graph such that edges within a group have a high weight while edges between different groups have a very low weight [White and Smyth, 2005].

Spectral clustering is a widely used graph clustering algorithm, based on a similarity graph, in various applications including exploratory data analysis, computer vision and speech processing. The basic idea for spectral clustering is to reduce the dimensionality of an (n, n) matrix to an (n, k) matrix by a Laplacian matrix L:

$$L = D - A \tag{2.2}$$

where D is the degree matrix where the degree of a vertex is the number of edges connected and A is the adjacency matrix where the row and column indices represent the vertices, and the entries represent the absence or presence of an

edge between the vertices.

The first step is to construct an affinity matrix where the value expresses how similar those vertices are to each other. There are several ways to construct the affinity matrix, for instance, k-nearest neighbour method and precomputed affinity matrix. The nearest neighbour approach assumes that k-close vertices should be in the same cluster. However, it is too sensitive of the choice of number of neighbours. Alternatively, precomputed affinity matrix is a matrix defined based on the similarity measurement between vertices, using a threshold on a distance metric-for example, euclidean distance, manhattan distance, minikowski distance, cosine similarity and jaccard similarity. After that, the objective is to minimise the weight of between-group connections.

Intuitively, for any distance matrix, 0 means identical elements and high values mean very dissimilar elements while high values in a similarity matrix means similar elements and 0 means significantly different elements. So any distance matrix could be transformed into a similarity matrix by applying the Gaussian kernel:

$$S_{matrix} = \exp\{-\frac{D_{matrix}{}^2}{2\delta^2}\} \tag{2.3}$$

where S_{matrix} means the similarity matrix transformed from distance matrix D_{matrix} and δ represents the width of the Gaussian kernel.

The main advantage of spectral clustering is that it does not make any assumptions about the cluster shapes. However, it is computationally expensive compared to partition based clustering, as pairwise similarities or distance must be computed, instead of only distances to centroids.

Example 1. Suppose there is a graph with 10 vertices, the pair-wise euclidean distance is already known and number of clusters is fixed to 2 in advance. The symmetric affinity matrix is shown as follows:

Based on table 2.1, it is difficult to tell the inter-vertex relationship. So we sort the matrix by row and column. Table 2.2 shows the closeness between vertices after sorting. Vertex A_0 is closer to vertex A_2 but far away from vertex A_6. Thus, vertex A_0 should be put into a group with A_2 but apart from A_6. When the Gaussian Kernel is used to measure the similarity, the matrix is transformed to that in table 2.3:

Table 2.1: Pairwise distance between 10 vertices

	Vertices name									
	A0	**A1**	**A2**	**A3**	**A4**	**A5**	**A6**	**A7**	**A8**	**A9**
A0	0.000	1.174	0.036	0.048	0.061	1.202	1.222	0.941	0.064	0.852
A1	1.175	0.000	1.151	1.180	1.175	0.971	0.821	0.992	1.171	0.894
A2	0.036	1.151	0.000	0.048	0.059	1.184	1.196	0.926	0.057	0.845
A3	0.048	1.180	0.048	0.000	0.055	1.205	1.229	0.942	0.043	0.851
A4	0.062	1.175	0.059	0.055	0.000	1.195	1.236	0.959	0.068	0.861
A5	1.202	0.971	1.184	1.205	1.195	0.000	1.150	0.993	1.204	1.131
A6	1.222	0.821	1.196	1.229	1.236	1.150	0.000	1.078	1.226	1.321
A7	0.941	0.992	0.926	0.942	0.959	0.993	1.078	0.000	0.944	0.901
A8	0.065	1.171	0.057	0.043	0.068	1.204	1.226	0.944	0.000	0.854
A9	0.853	0.894	0.845	0.851	0.861	1.131	1.321	0.901	0.854	0.000

Table 2.2: Distance matrix sorted by row and column

	Vertices name									
	A0	**A2**	**A3**	**A4**	**A8**	**A9**	**A7**	**A1**	**A5**	**A6**
A0	0.000	0.036	0.048	0.062	0.065	0.853	0.941	1.175	1.202	1.222
A2	0.036	0.000	0.048	0.059	0.057	0.845	0.926	1.151	1.184	1.196
A3	0.048	0.048	0.000	0.055	0.043	0.851	0.942	1.180	1.205	1.229
A4	0.062	0.059	0.055	0.000	0.068	0.861	0.959	1.175	1.195	1.236
A8	0.065	0.057	0.043	0.068	0.000	0.854	0.944	1.171	1.204	1.226
A9	0.853	0.845	0.851	0.861	0.854	0.000	0.901	0.894	1.131	1.321
A7	0.941	0.926	0.942	0.959	0.944	0.901	0.000	0.992	0.993	1.078
A1	1.175	1.151	1.180	1.175	1.171	0.894	0.992	0.000	0.971	0.821
A5	1.202	1.184	1.205	1.195	1.204	1.131	0.993	0.971	0.000	1.150
A6	1.222	1.196	1.229	1.236	1.226	1.321	1.078	0.821	1.150	0.000

Table 2.3: Similarity matrix obtained by Gaussian Kernel

	Vertices name									
	A6	**A5**	**A1**	**A7**	**A9**	**A8**	**A4**	**A3**	**A2**	**A0**
A6	1.00000	0.06626	0.20531	0.05260	0.01487	0.00015	0.00015	0.00013	0.00015	0.00014
A5	0.06626	1.00000	0.12821	0.09463	0.04084	0.00027	0.00028	0.00024	0.00026	0.00026
A1	0.20531	0.12821	1.00000	0.10076	0.09190	0.00036	0.00037	0.00032	0.00036	0.00034
A7	0.05260	0.09463	0.100769	1.00000	0.17349	0.00668	0.00661	0.00623	0.00667	0.00647
A9	0.01487	0.04084	0.09190	0.17349	1.00000	0.01698	0.01705	0.01598	0.01650	0.01639
A8	0.00015	0.00027	0.00036	0.00668	0.01698	1.00000	0.99017	0.99566	0.99049	0.99104
A4	0.00015	0.00028	0.00037	0.00661	0.01705	0.99017	1.00000	0.99247	0.98861	0.99113
A3	0.00013	0.00024	0.00032	0.00623	0.01598	0.99566	0.99247	1.00000	0.99240	0.99464
A2	0.00015	0.00026	0.00036	0.00667	0.01650	0.99049	0.98861	0.99240	1.00000	0.99548
A0	0.00014	0.00026	0.00034	0.00647	0.01639	0.99104	0.99113	0.99464	0.99548	1.00000

The values in table 2.3 shows the similarity of pairwise vertices. A large value means high level of similarity and small values indicates they are dissimilar. The highlighted red rectangle boxes show the two groups where intra-vertices are similar to each other but dissimilar to other vertices outside the boxes. If $\boldsymbol{k}$ is fixed to be 2, then vertex A_6, A_5, A_1, A_7 and A_9 should be put together and apart from another group [A_8, A_4, A_3, A_2, A_0].

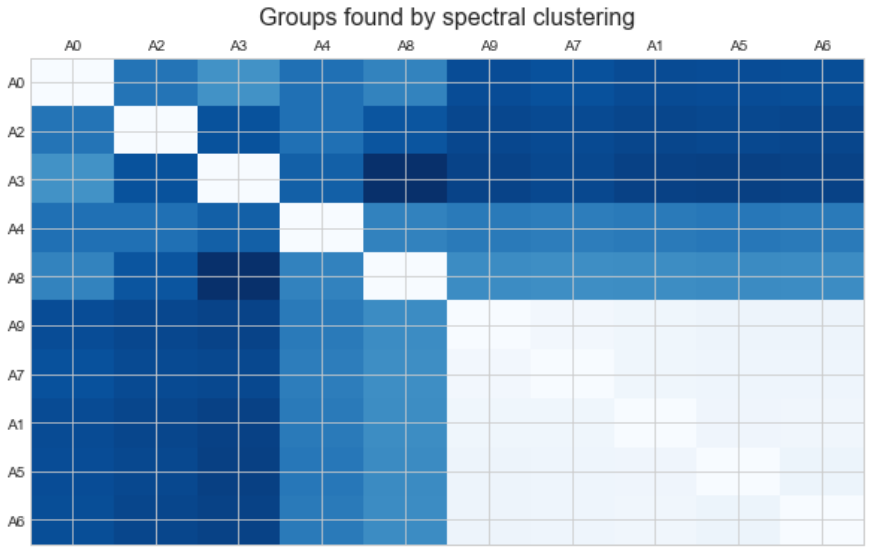
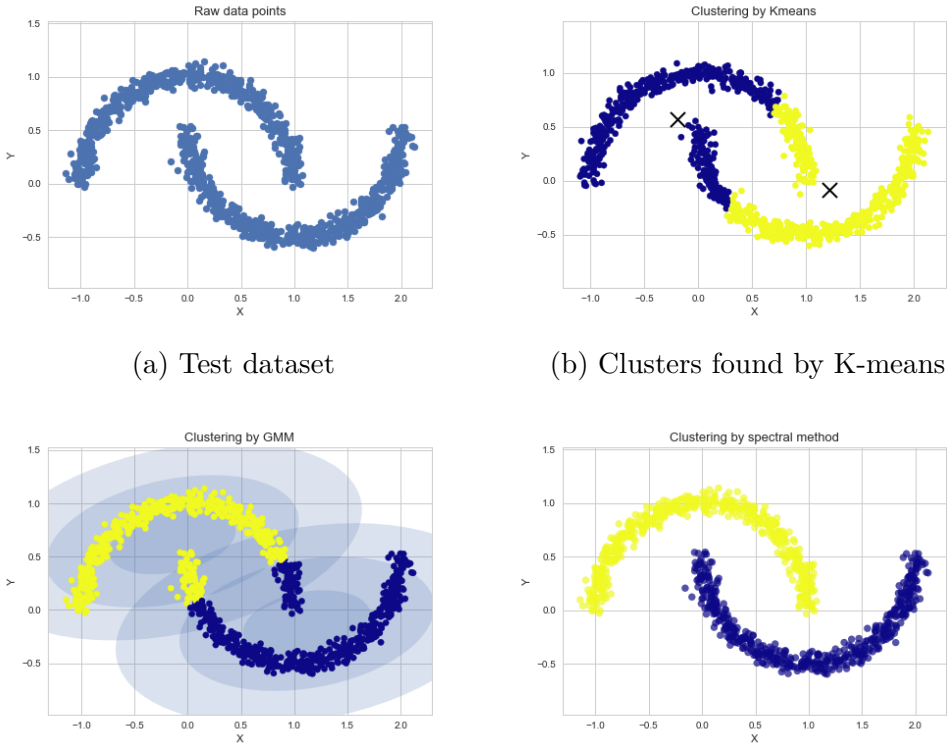


Figure 2.5: Groups found by spectral clustering

Figure 2.5 shows the groups found by spectral clustering where light colour means they are similar but deep colour means dissimilarity. When the number of groups are fixed to be 2, the groups found are the same group depicted in highlighted rectangle in table 2.3. Again, spectral clustering also suffer from the same problem as K-means and GMM: the number of clusters or groups has to be fixed in advance. In addition, the method to measure the similarity must also be specified, and can have a significant effect on the performance of this kind of clustering method.

Example 2. Figure 2.6 shows an example of K-means, GMM and spectral clustering where the number of clusters is fixed to be 2. The test data is two interleaving half circles. Since this shape is complicated, K-means and GMM may are not able to detect the right clusters. The affinity matrix of the spectral cluster is constructed based on the nearest neighbours method. Figure 2.6b and 2.6c show that neither K-means nor GMM can generate clusters that are similar to the original distribution. However, clustering obtained by GMM is better than that of K-means. Figure 2.6d shows that clusters obtained by spectral clustering is similar to the original distribution.



(a) Test dataset

(b) Clusters found by K-means

(c) Clusters found by GMM

(d) Spectral clustering

Figure 2.6: An example of K-means, GMM and Spectral clustering

2.1.3 Hierarchy Based Clustering

As mentioned before, we have to decide the number of clusters before using K-means or GMM. However, ideally, we would not be told how many clusters we should have as input. This is a gap hierarchical clustering bridges. It solves the problem of having to pre-define the number of clusters. For this kind of clustering, a hierarchical relationship among data is constructed.

There are mainly two types of hierarchical clustering: agglomerative hierarchical clustering and divisive hierarchical clustering. For agglomerative hierarchical clustering, initially each data point stands for an individual cluster. Then the closest two clusters are merged into a new cluster until there is only one cluster left. Divisive hierarchical clustering works in the opposite way. Instead of starting with n clusters, it starts with a single cluster and assigns all the points to that cluster. Euclidean distance is widely used as a distance metric to measure the closeness. Note that there are also some other distance metrics. Any distance measurement that is sensible for the problem should be considered. For instance, manhattan distance may be appropriate if clustering crime sites in a

city is demanded.

After selecting a distance metric, it is necessary to determine how the metric is to be applied. For example, it can be computed between the two most similar parts of a cluster (single-linkage) or the two least similar bits of a cluster (complete-linkage). In the past years, many linkage criteria have been developed. There are six popular methods to measure the closeness between two clusters:

1. Single, also known as nearest point algorithm, which assumes that the distance between two clusters is measured by the distance between the two nearest points.
2. Complete, also known as furthest point algorithm, which assumes that the distance between two clusters is measured by the distance between two farthest points.
3. Average, the distance between two clusters is defined as the average distance between each point in one cluster to every point in the other cluster.
4. Centroid, the distance between two clusters is defined as distance between centroids.
5. Median, also known as Weighted Pair Group Method with Arithmetic Mean, which assigns distance like the centroids method. When two clusters s and t are combined into a higher-level cluster u, the average of centroids s and t give the new centroid u. Then, its distance to another cluster k is simply the arithmetic mean of the distances between k and its members of u.
6. Ward, also known as the incremental algorithm. With this method, groups are formed so that the pooled within-group sum of squares is minimised. That is, at each step, the two clusters are fused which result in the least increase in the pooled within-group sum of squares.

Among these linkage criteria, Ward's method is widely used. Similar to K-means, this method aims to minimise the sum of squared distances of each observation from the average observation in a cluster and it matches the standard assumptions of how to compute differences between groups in statistics.

Example 1. Again, we use the same graph in spectral clustering as an example. Figure 2.7 shows the clustering result when incremental algorithm ward was used to measure the distance between groups. Vertex A_3 and A_8 are merged into a new cluster C_1 in the beginning because they are closest among all agents (distance

is 0.0666). Then vertex A_0 and A_2 are formed as a new cluster C_2. After that, vertex A_4 are merged with cluster C_1 to a new cluster C_3. This step will repeat until there is only one big cluster left.

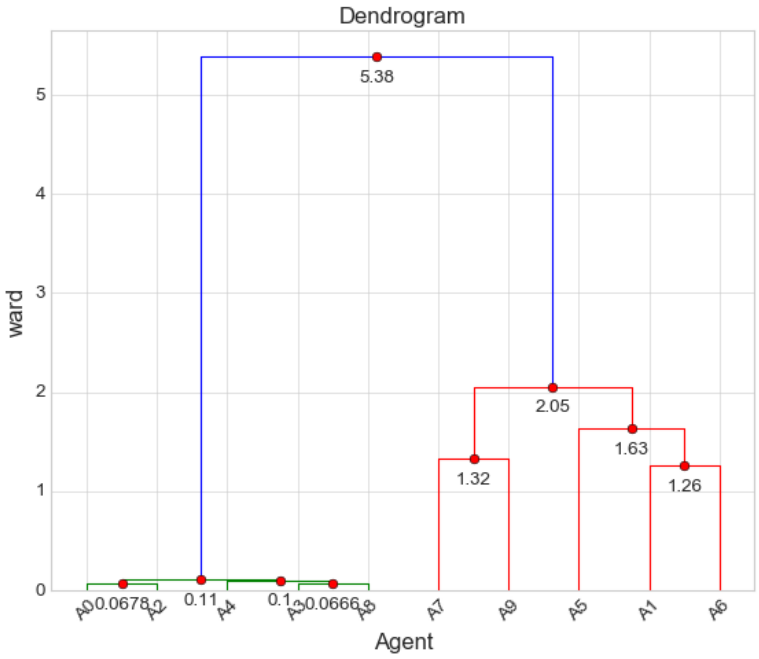


Figure 2.7: Dendrogram when ward's method is applied

Figure 2.7 shows a hierarchical relationship among clusters. Although the number of clusters are not required to pre-defined, a final number of clusters have to decide manually based on the hierarchical relationship. Note that the distance between the left cluster (where vertex A_3, A_8, A_0, A_2, A_4 are in) and right cluster (where vertex A_7, A_9, A_5, A_1, A_6 are in) is 5.38, two times longer than the second largest inter-cluster distance, which indicates that these two clusters are dissimilar and there might be 2 clusters.

To estimate the final number of clusters, a threshold distance could be defined manually such that two clusters are merged if the distance between them are below the threshold. Based on figure 2.7, we could set the threshold distance to around 3, then there are two final clusters.

2.2 Selecting the number of clusters

Note that all methods showed in previous sections requires that the number of clusters to be specified before start (except the hierarchical). Here we take the well-known iris flower dataset as an example. This dataset contains four

measurements of three different iris flowers. The measurements are: sepal length, sepal width, petal length, and petal width and the three types of iris are Setosa, Versicolour, and Virginica. Only the values of the petals' lengths and widths are used here. Figure 2.8 shows the differences between using two, three, four, and five clusters. The graphs also indicate that three is the most appropriate number of clusters. It makes sense when taking into account that there are three types of iris flowers in the dataset.

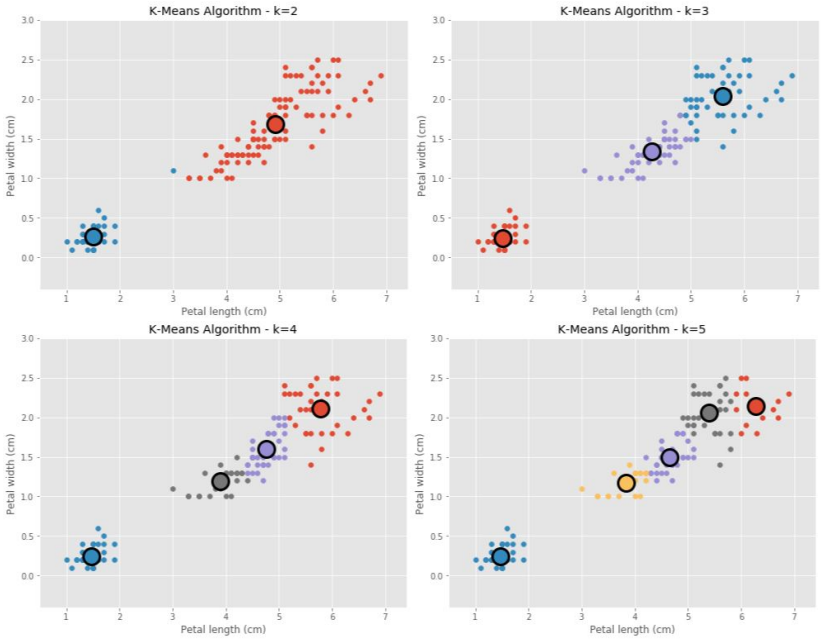


Figure 2.8: Clustering with different number of clusters on Iris dataset

However, there is no clear way to choose the appropriate number of clusters, since the shape and scale of the distribution of points in a dataset and the desired clustering resolution varies with applications. Note that increasing the number of clusters $\boldsymbol{k}$ will always reduce the amount of error (for example, Mean Squared Error), but the performance, usually measured by minimal intra cluster distance and maximal inter cluster distance, would be degraded when $\boldsymbol{k}$ is getting large until we reach the extreme case that $\boldsymbol{k}$ equals the number of data points. In this section, some frequently used methods to select the number of clusters are described.

2.2.1 Elbow method

The main idea for the elbow method is selecting the best number of clusters by calculating the sum of squares for a range of numbers of clusters and the selected number is then the one that had the steepest slope from the previous number

(the elbow point). When the number of clusters increases, data points are closer to the clusters centroids, so the sum of squared distance from each point to its assigned centroids will decrease. The improvements will decline, at some point rapidly, creating the elbow shape. This "elbow" point is a good indication that the underlying model fits best with that number of clusters. Algorithm 5.6 shows the main steps to pick the number of clusters by the Elbow method.

Algorithm 3: Elbow method to pick the number of clusters

1 **Input**: Range of number of clusters: $\boldsymbol{k} = \{k_0, k_1, \cdots, k_n\}$;
2 **Output**: The number of clusters that fits model best k_b;
3 **for** $i = 0; i \leq n$ **do**
4 Compute clustering algorithm for different values k_i ;
5 Calculate the total within-cluster sum of square distance (TWD);
6 Plot the curve of TWD for all different values k_i;
7 Find out the location k_b of "elbow" point in the plot;

The Elbow method is simple but it is not intended as a fully automated algorithm. The final step in algorithm 5.6 to determine the location is carried out by a human operator. Figure 5.6 below shows the best $\boldsymbol{k}$ is around 2, but could also be 3.

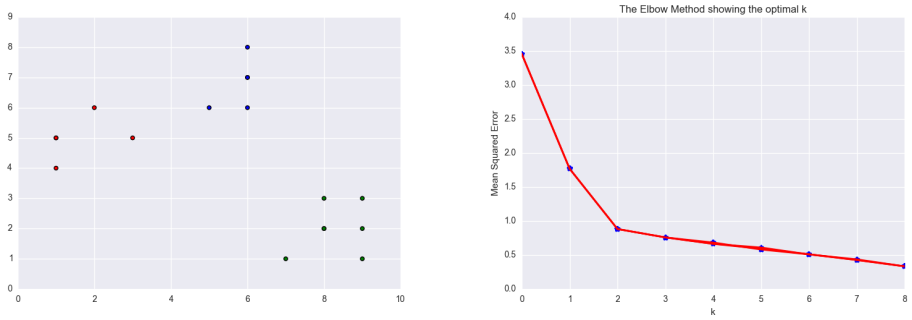


Figure 2.9: Test data and line chart with different number of clusters

2.2.2 Affinity Propagation

Frey and Dueck proposed a clustering algorithm called affinity propagation (AP) which does not require that the number of clusters to be fixed in advance. This method is based on message passing between data points [Frey and Dueck, 2007]. There are two kinds of messages exchanged between data points, responsibilities and availabilities. The responsibilities quantify how well-suited a data point x_i is to serve as the exemplar compared with other candidate exemplars and availabilities represent how appropriate it would be for data point x_i to pick another data point x_k as its exemplar, taking into account other points' preferences. Due

to its simplicity and general applicability, AP is getting more and more popular. However, the algorithm requires two inputs:

1. Similarities between data points, which represents how well-suited a point is to be another one's exemplar. The similarity matrix could be defined based on negative squared distance between points:

$$s(i,j) = -||x_i - x_j||^2 \tag{2.4}$$

   where a means average intra-cluster distance and b denotes the average distance to its nearest cluster.

2. Responsibility, representing each data point's suitability to be an exemplar.

Algorithm 4 shows the process of clustering method AP. The responsibility matrix and availability matrix will be updated until convergence. The convergence condition could be reaching an upper limit to the number of iterations or changes of the value before and after updating fall below a fixed threshold. After that, an element i will be assigned to an exemplar k which is not only highly responsible but also highly available.

Algorithm 4: Affinity Propagation

Input: Similarity matrix **S** and responsibilities matrix **R** ;
Output: Clusters that group similar data points together;
Initialise availability matrix **A**;
while *not converged* **do**
    Update responsibilities and availabilities as:
        $r(i,k) = s(i,k) - \max_{k' \neq k} a(i,k') + s(i,k')$
        $a(i,k) = \begin{cases} min\{0, r(k,k) + \sum_{i' \notin \{i,k\}} max\{0, r(i',k)\}\}, & \text{if } i \neq k. \\ \sum_{i' \notin \{i,k\}} max\{0, r(i',k)\}, & \text{if } i = k. \end{cases}$
Making assignment for data points i:
    $c(i,k) \leftarrow arg\max_k(r(i,k) + a(i,k))$

Figure 2.10 shows the clusters found by affinity propagation. Unlike the previous method, this method shows the clusters, rather than the estimated number of clusters. Compared to the elbow method and silhouette method (defined below), AP does not require that the number of clusters or range of number of clusters is fixed in advance. However, the computation cost is high since the availability matrix and responsibilities matrix must be updated each iteration.

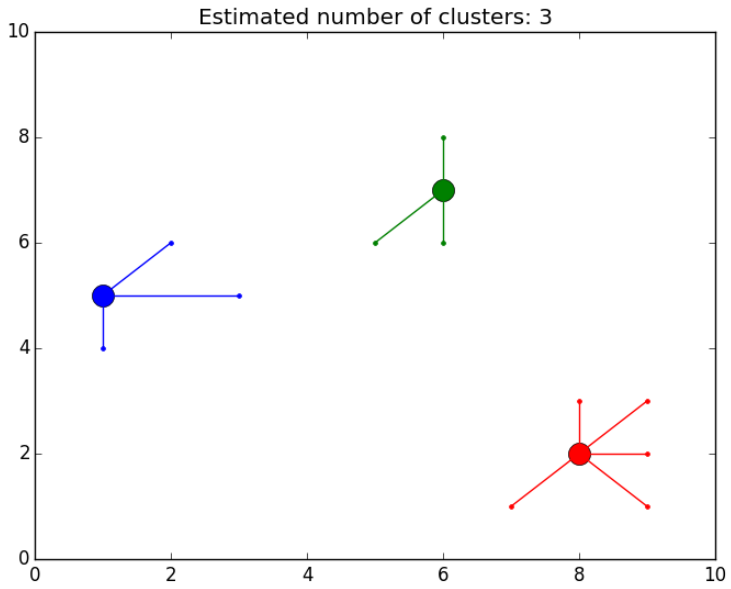


Figure 2.10: Estimated number of clusters obtained by AP method

2.2.3 Silhouette method

The silhouette value is a measure of how similar an object is to its own cluster compared to other clusters. The silhouette value ranges from -1 to $+1$, where a high value indicates that the object is well matched to its own cluster and poorly matched to nearest clusters. If most objects have a high value, then the clustering configuration is appropriate, otherwise, there may be too many or too few clusters. The silhouette is calculated as:

$$S_{index} = \frac{b - a}{max(a, b)} \tag{2.5}$$

where a means average intra-cluster distance and b denotes the average distance to its nearest cluster.

Algorithm 5 shows that the silhouette method is similar to the elbow method but the objective is different. Compared to the elbow method, the silhouette method can be automated, since it is simple to determine the maximum value [Kaufman and Rousseeuw, 2009]. However, the range of the number of clusters still has to be fixed in advance. Figure 2.11 shows the best $\boldsymbol{k}$ found by the silhouette method. The difference is obvious compared to figure 2.9. The average value of the silhouette increases to $\boldsymbol{k} = 3$, and then decreases sharply for $\boldsymbol{k}$ larger than

Algorithm 5: Silhouette method to pick the number of clusters

Input: Range of number of clusters: $\boldsymbol{k} = \{k_0, k_1, \cdots, k_n\}$;
Output: The number of clusters that fits model best k_b;
for $i = 0; i \leq n$ **do**
    Compute clustering algorithm for different values k_i ;
    Calculate the average silhouette (S);
Plot the curve of S for all different values k_i;
Find out the location k_b with maximum S in the plot;

3. The pronounced peak at $\boldsymbol{k} = 3$ indicates that this is the appropriate number of clusters. In this thesis, we apply silhouette method to estimate the initial number of clusters $\boldsymbol{k}$ for partition based clustering method, including K-means and GMM.

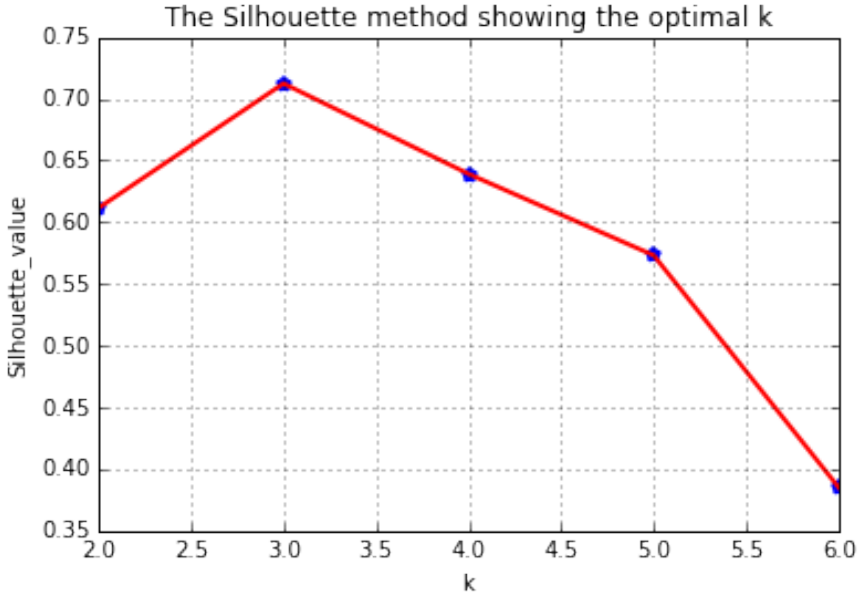


Figure 2.11: Number of clusters found by silhouette

Olatz et al. shows that Silhouette achieves the best results among the 30 cluster validity indices in most cases [Arbelaitz et al., 2013]. In this thesis, we use the Silhouette method to pick the number of clusters. It measures how similar a point is to its own cluster (cohesion) compared to other clusters (separation). The cohesion is measured based on the distance between all the points in the same cluster and the separation is based on the nearest neighbour distance. Formally:

$$Sil(C) = 1/N \sum_{C_k \in C} \sum_{x_i \in c_k} \frac{b(x_i, c_k) - a(x_i, c_k)}{max\{a(x_i, c_k), b(x_i, c_k)\}} \tag{2.6}$$

where

$$a(x_i, c_k) = \frac{1}{|c_k|} \sum_{x_j \in c_k} d_e(x_i, x_j) \tag{2.7}$$

$$b(x_i, c_k) = \min_{c_l \in \{C - c_k\}} \{\frac{1}{|c_l|} \sum_{x_j \in c_l} d_e(x_i, x_j)\} \tag{2.8}$$

2.3 Similarity measurement

In previous section, the similarity of two data points is represented as the distance between them. In this section, we also discuss the similarity in an agent level, which is used to predict whether an agent is different. The concept of similarity is important in almost every scientific field, such as mathematics, management and medicine. In this section, distance based, feature based and Earth Mover's Distance based methods to measure the similarity are discussed.

2.3.1 Distance based similarity

Distance is one of the most influential assumptions related to similarity: two objects can be said to be similar if the distance between their descriptions is small [Young and Hamer, 1994]. The two most popular distance measures are Euclidean distance and Manhattan distance. Given the coordinates of m observations A in a n-dimensional space $(x_{A1}, x_{A2}, \cdots, x_{Am})$, the Euclidean distance from observations A to some other observations B is:

$$d(A, B) = \sqrt{\sum_{i=1}^{n} (x_{Ai} - x_{Bi})^2} \tag{2.9}$$

The corresponding Manhattan distance distance is:

$$d(A, B) = \sum_{i=1}^{n} |x_{Ai} - x_{Bi}| \tag{2.10}$$

The distance then could be translated to a similarity via an exponential function [Shepard, 1987]:

$$s(A, B) = e^{-d(A,B)} \tag{2.11}$$

All types of distance obey certain properties, named the distance axioms. Generally, there are four distance axioms (take data points A, B, C as an example):

1. Non-negativity. $d(A, B) \geq 0$. If A and B differ, then distance $d(A, B) > 0$.
2. Equal self-similarity. $d(A, B) = 0$ if and only if $A = B$.
3. Symmetry. $d(A, B) = d(B, A)$.
4. Triangle inequality. $d(A, B) + d(B, C) \geq d(A, C)$. The distance between any two points can not be larger than the sum of their distance from a third point.

2.3.2 Feature based similarity

The validity of these distance axioms is criticised in some cases. For instance, Tversky et al. [Tversky, 1977] proposed an example against the symmetry axiom: most people judge the similarity of North Korea to China to be greater than that of China to North Korea. So a feature-based similarity was proposed to against the distance axioms, and it was defined as:

$$s(A, B) = \alpha g(A \cap B) - \beta g(A - B) - \gamma g(B - A) \tag{2.12}$$

where $g(A \cap B)$ denotes the features that are common to A and B, $g(A - B)$ denotes the features that are unique to A, and α, β and γ are predefined parameters. Based on the definition, it considered the common and distinct observations features and features in common increase the similarity while features that are unique to any observation decrease the similarity.

In many machine learning tasks, the definition of similarity plays a crucial role in the performance of proposed solutions. Distance based similarity is widely use in different areas, such as image retrieval and pattern recognition [Guo et al., 2002, Wren et al., 2006]. It provides a simple and mathematically convenient metric on raw features and an excellent survey on distance measure can be found in [Cha, 2007]. It is valid when features can not be defined, or when features consists of both continuous and categorical variables.

In some other areas, feature based similarity is widely used. For instance, in recommender systems, it was used to describe the preference [Agarwal and Chauhan, 2017, Han and Karypis, 2005]. Based on preference, it recommends products to a customer who are most likely to buy or music to a user who will find enjoyable. This feature-based recommend system has proven to be very successful in many

e-commerce settings, especially when user's needs and preferences are not clear defined.

2.3.3 Earth Mover's Distance based similarity

The Earth Mover's Distance (EMD) is a natural and intuitive metric to evaluate dissimilarity between signatures that are representations of two multi-dimensional distributions [Hitchcock, 1941, Rubner et al., 1998]. For cluster analysis, the distribution can be considered as a set of clusters where each cluster is represented by its mean or model. It measures how much work would have to be done modifying the two distributions so that they coincide exactly.

Given signature P = $\{(p_1, w_{p1}), ..., (p_n, w_{pn})\}$ and Q = $\{(q_1, w_{q1}), ..., (q_n, w_{qm})\}$, where p_i (or q_i) is the cluster representative and w_{pi} (or w_{qi}) is the weight of the cluster, and d_{ij} is defined as the ground distance between cluster p_i and q_j, the EMD between them is treated as a solution to a transportation problem. Treat elements in P as supplies and elements in Q as demands (corresponding W_{pi} and W_{qj} indicates the amount of supply and demand respectively). EMD is defined as the minimum work required to resolve the supply-demand transports, that find a flow F=$[f_{ij}]$ that minimises the overall cost:

$$WORK(P, Q, F) = \sum_{i=1}^{n}\sum_{j=1}^{m} f_{ij}d_{ij} \tag{2.13}$$

subject to the following constraints:

$$\begin{aligned} f_{ij} \geq 0 \quad & 1 \leq i \leq n; 1 \leq j \leq n \\ \sum_{i=1}^{n} f_{ij} \leq w_{pi} \quad & 1 \leq i \leq n \\ \sum_{j=1}^{m} f_{ij} \leq w_{qj} \quad & 1 \leq j \leq m \\ \sum_{i=1}^{n}\sum_{j=1}^{m} f_{ij}d_{ij} = min(\sum_{i=1}^{n} w_{pi}, \sum_{j=1}^{m} w_{qj}) & \end{aligned} \tag{2.14}$$

The first condition limits the flow to move from P to Q and not vice versa. The maximum amount of supplies and demand are restricted by P and Q in next two conditions, separately. The maximum work is limited by supply and demand in the final constraint. EMD is defined as the work normalised by the total flow:

$$EMD(P,Q) = \frac{\sum_{i=1}^{n}\sum_{j=1}^{m} f_{ij}d_{ij}}{\sum_{i=1}^{n}\sum_{j=1}^{m} f_{ij}} \tag{2.15}$$

The EMD has been used successfully in various real applications, such as image retrieval, key point matching and duplicate image identification [Rubner et al., 2000, Pele and Werman, 2008, Xu et al., 2008]. There are several advantages of using EMD over other dissimilarity measures: (1) robust to noise, (2) insensitive to topology change, and (3) globally shape-aware.

Although there are advantage of using EMD, it is hardly used outside the computer vision community due to its high computation cost. The time complexity is $O(n^3 logn)$ for a distribution that denoted by n signatures. To mitigate the high computation cost of EMD, there have been several proposals. Shirdhonkar and Jacobs [Shirdhonkar and Jacobs, 2008] transform the data array using a wavelet function and then an approximate EMD is obtained by the $L1$-norm of the difference of two transformed wavelet descriptors before computing the EMD distance directly. The computation time could be reduced to $O(n)$ by this method. Although the error, denoted by ratio of wavelet EMD to EMD or EMD to wavelet EMD, is bounded, it is shown that this error could be as high as 7.

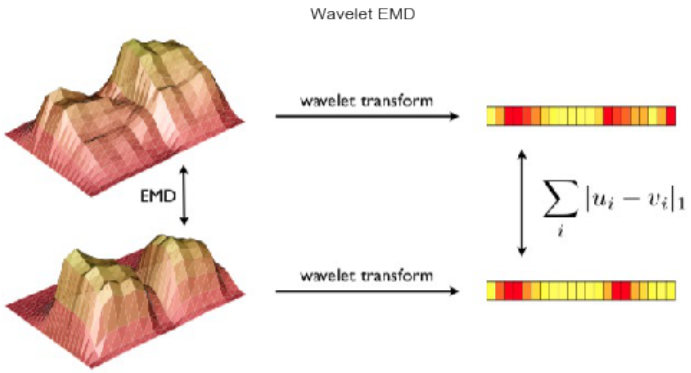


Figure 2.12: Wavelet EMD [Shirdhonkar and Jacobs, 2008]

Pele and Werman transferred the transportation problems to a transhipment problem by allowing the shipping of the goods to a intermediate destination, which could reduce the number of edges by an order of magnitude (reduced from $n^2 + N$ to $nK + n$ where n is the number of bins, N is the number of edges and K is the average number of edges going out) [Pele and Werman, 2009]. Besides, thresholded distance, rather than ground distance, is used (– see figure 2.13, where the striped yellow square is the new transhipment vertex). The computation time could be reduced to $O(n^2 logn)$ by this method. However, it is too complex for

the set-up process. In chapter 5 of this thesis, we will propose a simplification of EMD that is faster to compute, and which appears almost as effective when used in our clustering problems.

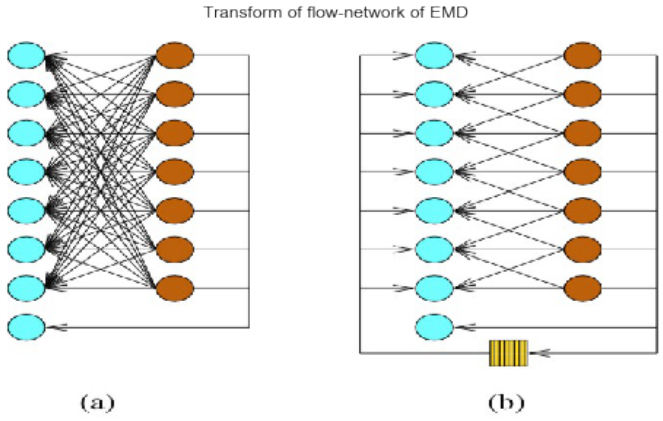


Figure 2.13: Robust EMD [Pele and Werman, 2009]

2.4 Information Dissemination in Wireless Networks

Information dissemination is one of most critical in-network operations for various applications, such as remote monitoring, assembly line monitoring and tracking. Under those circumstances, the network nodes have to spread or disseminate the information they have collected or sensed to other nodes via multi-hop paths. There are three widely used information dissemination protocols: 1) Flooding, 2) Gossip, 3) Sensor Protocols for Information via Negotiation (SPIN).

2.4.1 Flooding

Flooding protocol is one of the most widely used strategies to disseminate information in the network [Jetcheva, 2001]. A source node sends a packet to all of its neighbours. Each recipient node stores a copy of the packet and rebroadcasts the packet. This process continues until all the reachable nodes received the packet. This method of disseminating information is robust to node failures. However, flooding suffers from the following disadvantages [Tseng et al., 2002]:

1. Redundant rebroadcasts. In some cases, all neighbours of a specific node may have already received the message but that node still rebroadcast the message.

2. Channel contention. If many of its neighbours decide to broadcast the message in the same time, they have to compete for the channel.

3. Collision. Collisions are more likely to occur if too many neighbours decide to rebroadcast the message.

The redundant rebroadcast can be avoided by knowing the source ID and sequence number of each packet. Based on that, nodes can identify each packet and prevent broadcasting the same packet to the same node more than once. Since flooding is simple but effective, there are several approaches proposed in the literature to improve the efficiency, based on two considerations: reduce the possibility of redundant rebroadcasts and differentiate the timing of rebroadcasts [Tseng et al., 2002].

Based on the location information, the concept of a request zone is introduced to limit the search space for a desired route, by which the set of nodes to broadcast the packets is reduced [Sohrabi and Pottie, 1999]. Sabbineni et al.[Sabbineni and Chakrabarty, 2005] proposed a modified flooding data dissemination protocol for wireless sensor networks. The list of node IDs are stored in a special field of the packet header which tells the receiver nodes that all the nodes in the list already have the packet, hence it is unnecessary to forward the packet again. In addition, with the aid of GPS, a sensor can calculate its position in three dimensions. The monitored area is divided into virtual grids, and the ID of the grid in which the sender of the packet is currently in is also stored in the packet header. The gateway node will prevent the retransmission of packets the grid has already seen.

2.4.2 Gossip and weighted gossip algorithm

Rumours travel at a great speed and reach almost every member of the community in society without needing a central coordinator. Gossip algorithm is an asynchronous information exchange protocol inspired by this rumour style. In sensor networks, peer-to-peer networks or vehicular vehicles and social networks, providing communication is expensive in terms of time and resource. Gossip protocols try to solve the problem of flooding protocol, that is, disseminate a message to all the nodes in the network and each node sends the message to only a few of the nodes.

The basic idea behind it is simple. if a node wants to share some information to the other nodes in the network:

1. Periodically it selects randomly a node or a fraction of nodes and transmits

the information.

2. The node that receives the information updates its own records and then transmits the updated information.

3. Repeat until reach the stop criteria.

Based on the way nodes choose how many other nodes to communicate with, there are two main approaches to the gossip algorithm. One is called binary gossip, where each network node shares data with a single other node at each step. The other one called weighted gossip, where each node share data with all its network neighbours in a weighted fashion. Based on how nodes communicate with other nodes, the gossip algorithm could be classified as Push-pull and Push-sum gossip.

Algorithm 6: Push-pull binary gossip

```
1 Initialization: x_i;
2 while not stabilised do
3     Choose a neighbour node j uniformly at random ;
4     Send x_i/2 to node j;
5     x_i ← x_i/2;
6     if received pushing message from neighbour z then
7         Send pair x_i/2 to node z;
8         x_i = (x_i + x_z)/2;
```

Push-pull gossip is a synchronous protocol. For instance, in a push-pull based binary gossip (algorithm 6) , node i randomly chooses a neighbour j and halves its value ($\frac{x_i}{2}$) to node j (push). Upon receipt of pushing from node i, node j sends half of its value ($\frac{x_j}{2}$) and sends it back to node i (pull). node i and j update their value to $\frac{x_i+x_j}{2}$. Push-pull protocol does not require global knowledge or global coordination. However, the feedback required (pull) in this protocol may be difficult or even not exist anymore when the message is sent through a long route within the network.

Unlike push-pull gossip, push-sum gossip is an asynchronous protocol. At each time t, each node maintains a sum $s_{t,i}$ (initialized to $s_{0,i} = x_i$), and a weight $w_{t,i}$ (initialised to $w_{0,i} = 1$). At start of each time point , it sends pair $1/2(s_{0,i}, w_{0,i})$ to itself and a random neighbour. Each node i follows the protocol given as algorithm 7:

Weighted gossip is defined as the class of gossip-based algorithms following the sum and weight structure. The weight vector in weighted gossip is defined as $\mathbf{w} = \{w_1, \cdots, w_n\}^T$, which will be initialised to be all ones $\mathbf{1}$. Each node i

Algorithm 7: Push-sum binary gossip

Initialization: x_i, $\omega_i = 1$;
while *not stabilised* **do**
  Choose a neighbour node j uniformly at random ;
  Send $(\frac{s_i}{2}, \frac{\omega_i}{2})$ to node j;
  Let $\{(x_i, w_i)\}$ be all pairs sent to i;
  $s_i = \sum_i x_i$;
  $\omega_i = \sum_i w_i$;
  $x_i = \frac{s_i}{\omega_i}$;

choose a non-nagetive share $\alpha_{i,j}$ for each j, such that $\sum_j \alpha_{i,j} = 1$ and send an $\alpha_{i,j}$ fraction of its sum and weight to each j. The non-negative $n \times n$ transition matrix is defined as $\mathbf{K} = \{\mathbf{k}_1, \cdots, \mathbf{k}_n\}^T$.

Consider a undirected graph G=(V, E), where V is the set of vertices and E denotes the edge set. We denote the vector of the values in all nodes $\mathbf{x} = \{x_1, \cdots, x_n\}^T$. The global mean value $\overline{x} = \frac{1}{n}\sum_i^n x_i$. The weight vector in weighted gossip is defined as $\mathbf{w} = \{w_1, \cdots, w_n\}^T$, which will be initialised to be all ones $\mathbf{1}$. Each node i communicates with its neighbours to exchange messages and hold a codebook maintaining the share probability to its neighbours denoted by $\mathbf{k}_i = \{k_{i,j} | (i,j) \in E\}^T, s.t. \forall k_{i,j} \geq 0, \sum_j k_{i,j} = 1$.

Algorithm 8: Push-sum Weighted gossip

At node i ;
Input: The initial value $x = \{x_0, x_1, \cdots, x_k\}$ and $\omega = \{\omega_0, \omega_1, \cdots, \omega_k\}$;
Set of neighbours of node i is denoted as $\Gamma^{(i)}$, and $|\Gamma^{(i)}| = N_{nei}$;
β=0.01 ;
Output: Average value $x = \{\bar{x}, \bar{x}, \cdots, \bar{x}\}$, $|x| = K$, where $\bar{x} = \frac{\bar{s_i}}{\bar{\omega_i}}$;
$\alpha_{i,j} = \frac{1}{1+N_{nei}}$, $\forall j \in \Gamma^{(i)}$;
while *the stop condition is not satisfied* **do**
  for $j \in \Gamma^{(i)}$ **do**
    send the weighted pair $(\alpha_{i,j}x_i, \alpha_{i,j}\omega_i)$ to node j ;
  // update process ;
  Let $\{(s_j, \omega_j)\}$ be all pairs sent to node i ;
  $\widehat{s_i} \leftarrow \sum_j s_j$;
  $\widehat{\omega_i} \leftarrow \sum_j \omega_j$;
  $\bar{s_i} = \alpha_{i,j}s_i + \widehat{s_i}$;
  $\bar{\omega_i} = \alpha_{i,j}\omega_i + \widehat{\omega_i}$;

To summarise, Gossip algorithm is robust, scalable, fault-tolerance and decentralised. However, it is not efficient since redundant information can arrive several

times to a node due the randomness of picking neighbours to share information. More importantly, Gossip is not efficient in time, and is not suitable for scenarios where there is limited time to propagate the information [Birman, 2007].

2.4.3 SPIN

SPIN is a negotiation based information dissemination protocol [Heinzelman et al., 1999]. The main idea behind SPIN is sharing descriptors with neighbours before transmitting actual data. When a description is received, a node determines whether it has the data or not. If this data is new, a request message is sent to the source, which replies with the actual data. Thus, the classic problems of flooding such as redundant rebroadcast, channel contention and collision are solved by this negotiation, improving energy efficiency. An excellent survey highlighting the issues in SPIN protocol and possible enhancement of SPIN protocol is presented in [Patel and Parekh, 2014].

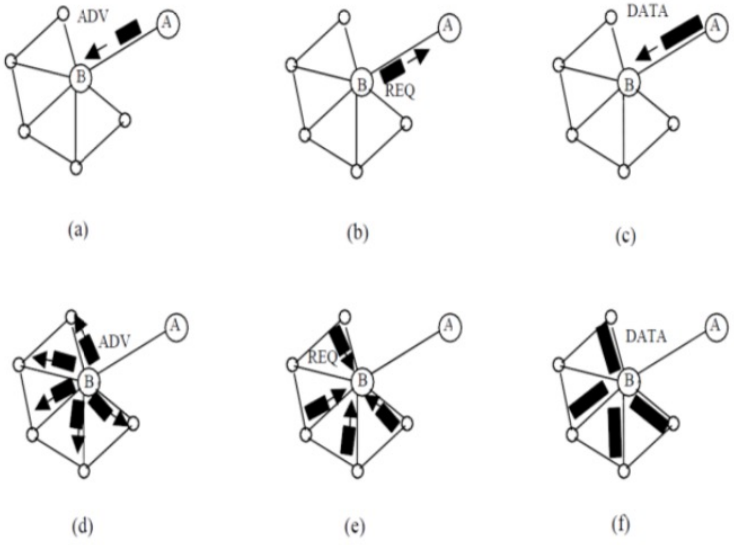


Figure 2.14: SPIN protocol [Patel and Parekh, 2014]

Figure 2.14 shows an example how SPIN works in the network. There are three different defined message: 1), ADV message: advertise a particular meta-data before transmitting actual data, 2) REQ message: request the specific data, 3) DATA message: message that carry the actual data. Node A initialise a ADV message to node B. When received this ADV message, B responds by sending a request REQ to node A. Node A send the DATA message after receiving the REQ. Then node B sends out ADV message to all its neighbours, neighbours who has not seen the message send a request to node B.

2.5 Model selections

In this section, we describe two different methods to perform model selections. The first method is information-theoretic criteria (IC): Akaike information criterion (AIC) and Bayesian information criterion (BIC). They are both penalised-likelihood criteria, and are defined as:

$$AIC = -2logL + 2p \tag{2.16}$$

$$BIC = -2logL + logn * p \tag{2.17}$$

where L is the likelihood function and p is the number of parameters in the model. Both Akaike information criterion (AIC) and Bayesian information criterion (BIC) are likelihood function based criterions for model selection among a finite set of models. AIC estimates the relative distance between fitted likelihood function of the model and unknown true likelihood function of the observations, so that a lower AIC means a model is considered to be closer to the truth. By trying different number of components, we can estimate the optimal components. BIC, by the name, is an estimator of a posterior probability function of a model being true, so that a lower BIC means that a model is considered to be more likely to be the true model. When fitting models, it is possible to increase the likelihood by adding parameters, but doing so may result in overfitting. Both BIC and AIC attempt to resolve this problem by introducing a penalty term for the number of parameters in the model. Since this method is used to detect single pattern, we assume that if estimation by any method, AIC or BIC, is a single pattern, we confirm that there is a single pattern.

As well as information-theoretic criteria (IC), P-value is also used to perform model selections in statistics. When there is little data available or we have no idea about the standard error for a given observation, a sampling technique, for instance, bootstrap, can be used to estimate the true population of given observation. Draw n observations from the sample with replacement and calculate the sample mean $\bar{X}_i$. These sample means form the bootstrap sampling distribution. Repeat this over and over (typically 1,000 or 10,000 times) to end up with $\mathbf{B}$. The central limit theorem says that the distribution of bootstrap sampling will be closely approximated by a normal distribution and the mean of this distribution will be very close to the true mean. The P-value is calculated by the confidence interval (CI) as follows [Altman and Bland, 2011]:

$$P = e^{-0.717Z-0.416Z^2} \tag{2.18}$$

where test statistic Z is defined as estimation Est divided by standard error SE with $SE = (U - I)/(2 * 1.96)$, and U and I are the upper and lower limits of a 95% CI respectively.

So which method is better for model selection? Both P-value and IC are different approaches to mine meaningful information from data. P-value is based on sampling theory while AIC is based on information-theoretic criteria. AIC is used to compare competing models for adequacy. The model with the least IC is considered as the most adequate. P-values of parameters indicate significance of a predictor by testing hypotheses that regression coefficients differ from zero (hypotheses means the number of component here). Using p-values in stepwise regression to select hypotheses is definitely not recommended since looking at individual p-values can be misleading [Xin and Zhu, 2012]. Large p-values could be caused by the fact that there is high correlation between variables. This does not mean the variables are useless. As a quick rule of thumb, selecting model with the AIC criteria is better than looking at p-values.

2.6 Distributed learning frameworks for wireless networks

We are interested in learning the wider network through message exchange when each agent in the network only see a fraction of the data points [3]. Distributed learning methods are proposed to avoid the privacy, security and overhead issues caused by centralising all raw data to a central server or cloud. Generally, there are two different distributed learning frameworks for power-constrained wireless networks: distributed learning with a fusion centre and with in-network processing. Figure 2.15 shows the frameworks, where $S_i = \{(X_{ij}, Y_{ij})\}$ represents a stationary sensor's reading.

When a fusion centre is available, sensors communicate directly with a fusion centre and it focus on how learning is effected when gathering all data is impossible for the communication constraints. Sending all collected data to the fusion centre

[3] Hadoop spreads the data out, and then gathers the result, so it is different [Greenhill and Venkatesh, 2007, Kotary and Nandal, 2019, Corbett et al., 2013]

is a direct way but is costly in terms of energy and bandwidth. Alternatively, designating a small subset of nodes to send data is more principled method. For instance, partitioning the sensors into sub-groups and electing a cluster head for each sub-group. The selected cluster head is required to send data to the fusion centre. This setting is relevant to WSN whose primary purpose is data collection.

The second framework is formed with an ad hoc structure in the right side of figure 2.15. The typical assumption is that the topology of the network is dynamic and might be unknown prior to deployment, but sensors are highly independent and able to make their own decisions. The goal is to learn a wider network pattern. Now the question is that how to achieve collaborative learning by efficient inter-sensor communication and in-network processing such that each sensor derives a global estimation? Note that the global estimation is learned through sharing information and built up gradually in an ad hoc way where all sensors contributed a little bit. The global estimation learned may vary with nodes and further work has to be done if it is required that all nodes learned the same global picture.

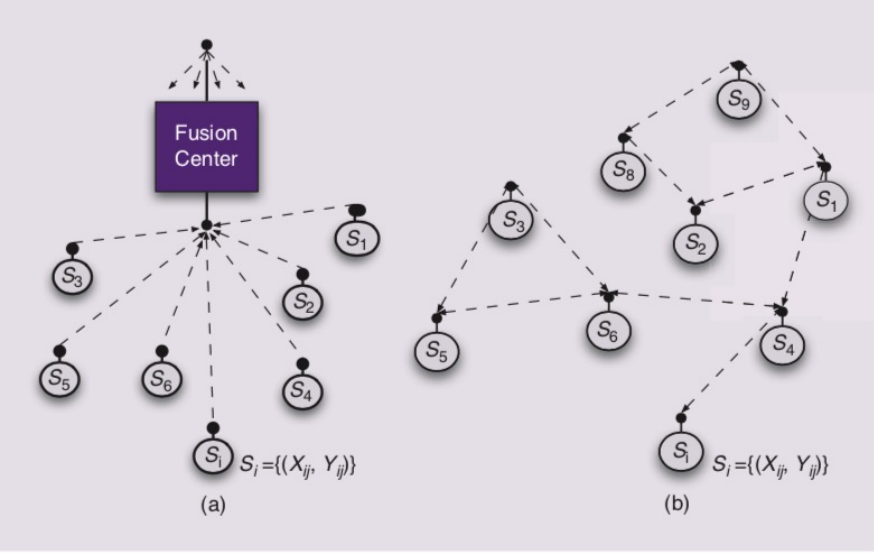


Figure 2.15: Distributed learning framework: (a) with a fusion center. (b) with in-network processing [Predd et al., 2006]

Clustered approaches, which differs from the clustering algorithm in section 2.1, has been frequently used in distributed learning with a fusion centre [Bandyopadhyay and Coyle, 2003, D'Costa et al., 2004, Nguyen et al., 2005]. In section 2.1, clustering algorithms are applied on data points owned by individual sensors. Nodes in the network are grouped together as clusters in clustered approaches. The nodes are divided into groups and one sensor of each group is assigned as cluster head. Communication inside clusters could be efficient since nodes in a

cluster are nearby. The cluster head filters the data received from nodes within its group and send the fusion centre a summary that best describe the locally learned rules. However, the cluster head election is still a challenge in the literature, and this learning framework is not suitable when topology of network is unknown in advance.

The in-network process usually comprises: 1) local learning step where each node defined a summary that best describe the locally learned pattern; 2) an communication step where nodes exchange information with their neighbours; 3) an update step where each node refines its local estimation based on information received from neighbours; and 4) a converge step where agents stop updating their local estimation when certain conditions, for instance, a finite number of iterations, are met. In the following sections, we describe the state of the art related to these steps involved in the in-network processing.

2.6.1 Local learning

Traditional machine learning (ML) approaches are designed to learn from a unique dataset, where components of dataset are centralised in a single database or location. When those technologies are applied to distributed data, collection of data in a central database is normally required, which is usually either ineffective or infeasible due to high transfer cost and privacy issues [Peteiro-Barral and Guijarro-Berdiñas, 2013]. This motives learning from distributed data in an effective and efficient way: local learning is carried out in each location independently and local models are then aggregated to the global model. Distributed data learning is highly scalable in amount of data, respects the privacy of components that spread out over multiple locations and reduces the high cost to centralise raw data.

In this thesis, we focus on unsupervised learning. As mentioned before, clustering is frequently used in distributed learning in wireless networks and there are several approaches to define the clusters. Clusters could be defined by a hierarchical method, where the clusters are decided by means of agglomerative approaches that recursively merge and split clusters until a stopping criterion is satisfied [Yin et al., 2017]; or represented by centroids, which is the value of all members with in this group on average [Krishnamachari et al., 2002]; or areas of high density in the density-based clustering [Birant and Kut, 2007]. In spectral clustering, clustering are performed in a transformed space [Scardapane et al., 2016]. Finally, in the distribution-based clustering, clusters are defined as distribution (mean

and standard deviation) that best model all objects belonging to in a probability perspective and GMM is the most adopted approach to describe clusters [Moon, 1996].

K-Means was proposed for local learning in large-scale peer-to-peer wired networks [Datta et al., 2008, Di Fatta et al., 2013, Bendechache et al., 2016, Bénézit et al., 2010]. The k-means algorithm is distributed by inserting an exchange of messages between neighbouring nodes after each internal iteration of the basic K-means loop (the selection of centroids, and the assignment of individual data points to each centroid) on the node's local data. After assigning the data points, each node transmits a description of its centroids and the number of data points associated with them to its neighbours. In [Tsiligaridis and Flores, 2016], local data was model by GMM instead, where each node estimate the means and the covariances of the clusters independently.

Recently, to fully utilised the shape information of clusters, Bendechache et al. [Bendechache et al., 2016] represent each cluster by its boundary points as well as centroids, and then exchange the boundary and the number of internal points. Figure 2.16 shows an example to obtain the boundary points by Delauney triangulation. Similarly, Kotary et al. [Kotary and Nandal, 2019] proposed that the using of Euclidean distance based cluster assignment make K-means unable to detect the clusters that have arbitrary shape and a symmetric distance was used instead to assign data points. Cluster centroids and Lagrangian multipliers of the diffusion method are shared among neighbours.

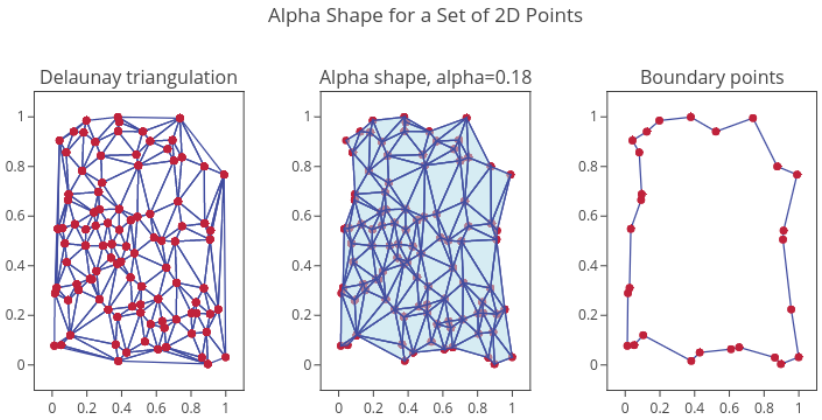


Figure 2.16: An example of obtaining boundary points of a cluster

PCA is another widely used unsupervised method used for feature extraction in

wireless network [Jiang et al., 2011, Zhou et al., 2014, Yu et al., 2017]. It identifies and keeps the highest priority information from the feature space to reduce the dimensionality. Given the number of principle components to find, the output of PCA is a linear combination of orthogonal principal components. In WSNs, PCA is used to reduce the dimensionality of the data either at sensor level or at cluster head level to reduce the communication overheads. Challenges, such as localization, fault detection, data aggregation, and target tracking, have adopted PCA.

Dictionary learning is another widely used method in many applications including image denoising, dimensionality-reduction, feature-extraction, and novel document detection [Mairal et al., 2010, Shen and Huang, 2008, Mairal et al., 2009]. It chooses a sparse matrix, called dictionary, to represent the data. The basic idea is that, given T samples of a P-dimensional data sequence, $\boldsymbol{Y} = [\boldsymbol{y_1}, \cdots, \boldsymbol{y_T}] \in \boldsymbol{C}^{P\times T}$, there must exist a set of K atoms or a dictionary, $\boldsymbol{D} = [\boldsymbol{d_1}, \cdots, \boldsymbol{d_k}] \in \boldsymbol{C}^{P\times K}$, such that each data sequence is a linear combination of only a few atoms, $\boldsymbol{Y} \approx \boldsymbol{DX}$where $\boldsymbol{X}$ has sparse columns. A basic dictionary can be obtained by using a PCA. Learning the representation of the data is useful for many tasks such as storing, transmitting or analysing the data to understand its content.

Distributed Dictionary Learning have proposed to adapt decentralised scenarios [Chainais and Richard, 2013, Chen et al., 2014, Daneshmand et al., 2019]. The idea is that each agent learned the dictionary $\boldsymbol{D_n}$ and corresponding coefficients $\boldsymbol{X_n}$ on its own data. The choose of coefficients $\boldsymbol{X_n}$ is also known as inference problem, that infer a matrix that as sparse as possible and the difference between $\boldsymbol{D_n}$ $\boldsymbol{X_n}$ and real vector $\boldsymbol{Y}$ is minimised. Then agent communicate with neighbours. Eventually, all agents will learn the global dictionary and the raw data could be recovered by global dictionary multiply a matrix called coefficients $\boldsymbol{X}$.

Based on the contents shared, there are mainly two different types of dictionary learning methods. The first one is sharing dictionary with others. Chainais et al [Chainais and Richard, 2013] proposed a diffusion-based method to learn a common dictionary over a sensor network where local dictionary was shared among neighbours. Upon receipt of new dictionary, an Adapt-Then-Combine (ATC) strategy is used to update its estimates by averaging neighbouring estimates. For the privacy issue, dual variables are shared instead in [Chen et al., 2014], and diffusion strategies are applied to solve the inference problem by considering dual function of the inference problem as an aggregation of individual cost functions associate with different agents.

Dictionary learning is useful because it significantly reduces the dimension of the raw data. However, the resulting dictionary learned is a dense matrix, and its manipulation can be computationally costly both at the learning stage and later in the usage of this dictionary. For this issue, dictionary learning is only applied to relatively small-scale problems [Le Magoarou and Gribonval, 2015].

2.6.2 Information exchange and data aggregation

In-network communication is a necessary step for further model updating in the in-network processing procedure. The architecture of the network plays a vital role in the information exchanging and updating process. In this section, we describe the communication approach and data aggregation protocol together under different structures. Depending on the network structure, there are several types of widely used data aggregation methods, such as Flat network, Cluster-based, Tree-based, Chain-based and hybrid aggregation protocols. There is a review of various energy efficiency data aggregation schemes in [Rajagopalan and Varshney, 2006].

Cluster-based topology: In this approach, network is divided into a number of clusters. Each cluster has a cluster-head which is responsible for communication with a sink node or base station. Merging the data belonging to a single cluster is an important way to reduce the energy consumption.

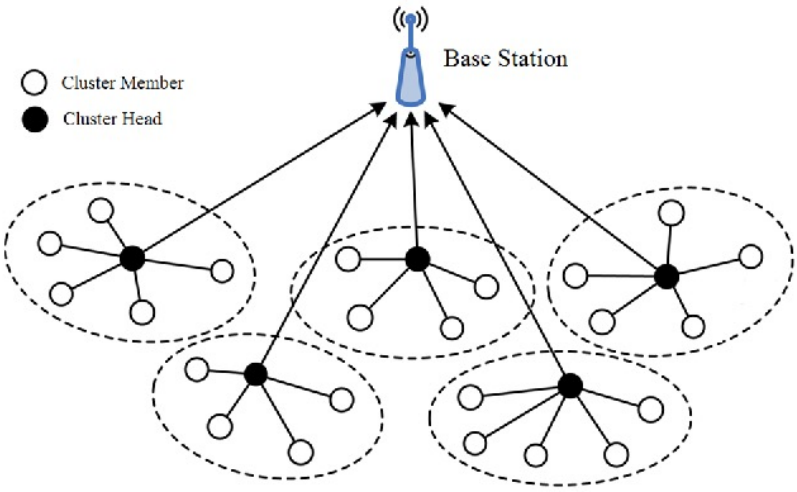


Figure 2.17: An example of Cluster-based topology [Gui et al., 2016]

The Low-energy adaptive clustering hierarchy (LEACH) is a widely used protocol in WSN to lower the energy consumption of network [Heinzelman et al., 1999]. Since acting as a cluster head requires more consumption of energy, the election of a cluster head is based on residual energy. All node that are not cluster heads

only communicate with the cluster head. There are a lot of extensions to address the limitations of LEACH, such as scalability and the need for all nodes to be continuously listening [Zhang et al., 2010, Heinzelman et al., 2002, Lindsay et al., 2001, Yueyang et al., 2006].

Chain-based topology In some cases, if the cluster head is far away from the sensors, it may take excessive energy in communication. Alternatively, each sensor transmits only to its closest neighbour. Chain-based topology is based on this idea (figure 2.18). A node receives data from one of its neighbours, forward the data to its other neighbours along the chain. Eventually the leader node (selected by some methods), which is similar to cluster head, transmits the aggregated data to the sink. However, the formation of chain normally requires that all nodes have global knowledge of the network.

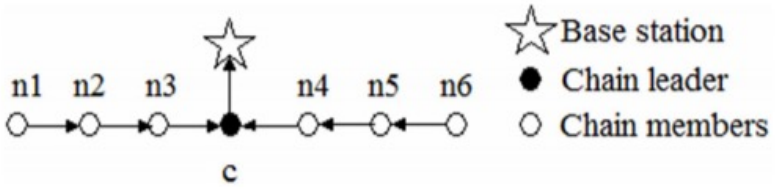


Figure 2.18: An example of Chain-based topology

Tree-based topology: In a tree-based network, sensor nodes are organised into a tree and intermediate nodes, except the root node, performs data aggregation when received information from its child node. Eventually, the root node will receive all information and produce the final model. For this kind of structure, constructing an energy efficient tree is widely explored [Ding et al., 2003, Tan and Körpeolu, 2003].

Bendechache et al. [Bendechache et al., 2016] represent each cluster by its boundary points, and then exchange the boundary and the number of internal points. Messages flow up the tree to the root. When a node receives multiple descriptions, it computes a new description by merging any overlapping clusters into a single cluster, and computing the new boundary. The algorithm is initialised with a much higher $\boldsymbol{k}$ value than is expected, to allow this cluster merging to reduce the number of clusters. This approach solves the issue of arbitrary shapes, but introduces failure points at each aggregation, and requires a secondary communication from root to leaves to disseminate the final clusters. It also exposes some of the raw data to neighbouring nodes, in order to specify the cluster boundary.

Hybrid topology: A hybrid topology comprises two or more structures [Zhang

et al., 2010, Liang et al., 2008, Hong et al., 2008]. Chen et al. [Zhang et al., 2010] proposed WST-LEACH algorithm. By taking the remaining energy, distribution density of nodes and distance from cluster heads to the base station into consideration, it first group local nodes into clusters. The clustering protocol is based on well-known LEACH protocol. Nodes within the cluster transmit to cluster head, and cluster head aggregate the data and forward it. Then a spanning tree is created by using cluster heads. After that, cluster heads sent the aggregated data to the base station via tree. Similarly, a Cluster-based Minimal Spanning Tree is created to maximise the network lifetime [Liang et al., 2008]. The cluster head is elected based on residual energy. After that, the cluster heads create a routing tree simultaneously and finally the base station receives data from cluster heads.

Flat topology: When topology of network is flat, nodes usually transmit information to their neighbours, which is determined by the transmission range. Sensors who receive information will update their local model. In [Lin et al., 2018], a double-clock consensus-based K-means algorithm was proposed for time-varying topology, where agent could only exchange size and summation of all data points with their one-hop neighbour. Total communication cost and number of iteration are considered. However, the cluster number k is assumed to be known for all agents.

Distributed K-Means [Datta et al., 2008] was proposed for use in large-scale peer-to-peer wired networks. The k-means algorithm is distributed by inserting an exchange of messages between neighbouring nodes after each internal iteration of the basic K-means loop (the selection of centroids, and the assignment of individual data points to each centroid) on the node's local data. After assigning the data points, each node transmits a description of its centroids and the number of data points associated with them to its neighbours. Once a node has received an update from all of its neighbours, it proceeds to the next round. It assumes centroids are listed in a specified order, and computes a weighted average of the centroids, which it then uses as the initial centroids for the next iteration. The algorithm terminates after successive updates do not change the centroids beyond a specified threshold. In this scheme, no raw data is exchanged, thus satisfying the basic privacy requirements, and the centroid summaries reduces the amount of data being transmitted. However, the algorithm requires synchronisation, which in wireless networks may introduce significant delay, and may fail to converge if one node has initial data whose distribution is significantly different from the average. The exchange of just centroids and counts also discards information

about the shape of the cluster. When clusters have significantly different shapes, this may again cause problems for convergence.

Gossip-based methods for distributed K-means are proposed in [Di Fatta et al., 2013, Bénézit et al., 2010]. Rather than sharing message with all its neighbours, each node selects a small subset of its neighbours to communicate with at each round, and includes a damping factor in the weighted average to avoid oscillations. Each node selects a small subset of its neighbours to communicate with at each round, and includes a damping factor in the weighted average to avoid oscillations. Although global synchronisation is avoided, local synchronisation is still required. In practice, the nature of the gossip algorithm means that many rounds are required before convergence. Since there is no global synchronisation, nodes are unaware of when other nodes have converged, and so another algorithm running in parallel is required to detect termination, or the algorithm is forced to terminate after a fixed number of rounds.

Compared to hierarchical networks, such as hybrid topology, cluster-based topology and tree-based topology, the flat network dose not suffer from the overhead involved in cluster or tree formation throughout the network. Besides, the network may still be operational even some nodes fail. For these reasons, flat topologies have been widely used for distributed learning in wireless network [Lin et al., 2018, Datta et al., 2008, Bendechache et al., 2016, Di Fatta et al., 2013, Bénézit et al., 2010].

Data aggregation function: Each of the different topologies can use different data aggregation functions and they can be summarised as follows: duplicate suppression and other aggregation functions with multiple inputs (for instance, sum, mean, max, min, weighted mean and weighted sum) [Krishnamachari et al., 2002, Giridhar and Kumar, 2005]. The aggregation function computes the relevant summary of the data. Taking temperature reading in environmental monitoring as an example, the final estimated temperature may be the mean temperature of all readings, or median, or mode while the maximum temperature reading might be much more interesting in alarm networks.

Figure 2.19 shows an example of tree-based aggregation process, where the arrow indicates the routing path and $f(-)$ is the data aggregation function. Since node 1 and 2 are leaf nodes, they will simply transmit the information (raw or processed) to their parent node 3. Node 3 will combine a and b with its own data c by a aggregation $f(-)$, and $f(a, b, c)$ will be sent to its parent node 5. This process will be repeated until all aggregated information reached the sink

node. If the aggregation function $f(-)$ is to compute the mean of all readings $i = \{a, b, c, e, x, y, g, z, m\}$, the mean h should be:

$$h = \frac{a+b+c+e+x+y+z+g+m}{9} \tag{2.19}$$

If the data aggregation function mean is carried out in intermediate nodes, the final reading $\bar{h}$ is computed as:

$$\begin{aligned} \bar{h} = f(j,k,m) &= \frac{m + f(d,e,g) + f(x,y,z)}{3} \\ &= \frac{m + f(x,y,z) + (g+e+f(a,b,c))}{3} \\ &= \frac{m + \frac{1}{3}(x+y+z+g+e) + \frac{1}{3}(a+b+c)}{3} \\ &= \frac{3m + x + y + z + g + e + \frac{1}{3}(a+b+c)}{9} \end{aligned} \tag{2.20}$$

Which is approximately equal to the actual reading h based on the assumption that readings in that monitoring area are similar. Note that if the data distribution is not uniform, then this might be significantly wrong, because it gives 9 times as much weight to m as it does to a, b or c.

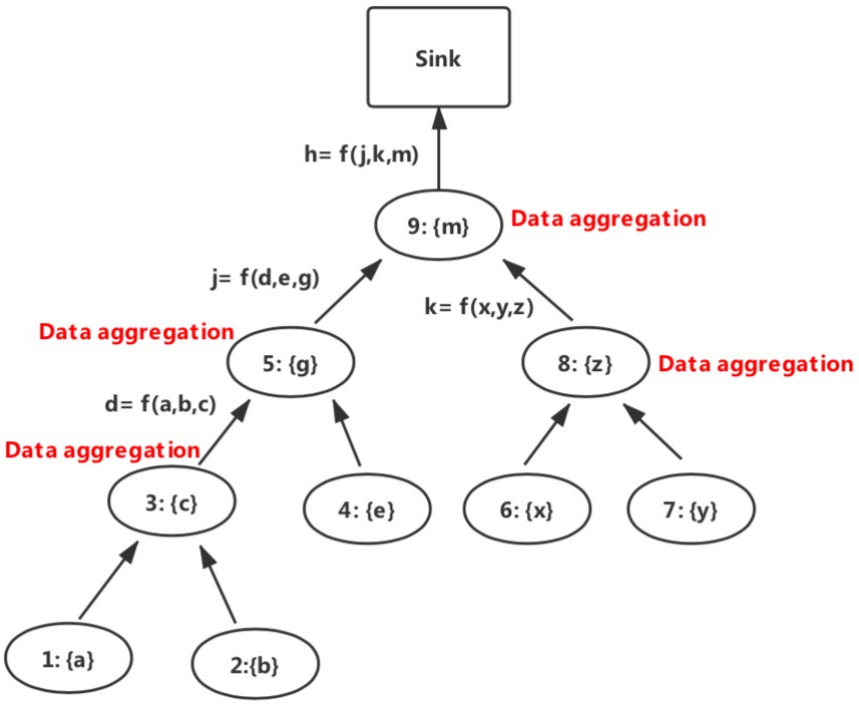


Figure 2.19: An example of tree-based aggregation process

In [Datta et al., 2008], a weighted average is used for the aggregation function.

After a sensor received centroids and counts from its neighbours, it updates its own information using the average weighting scheme (algorithm 9). Similarly, Gossip-based aggregation methods update their information using a weighted average [Di Fatta et al., 2013, Bénézit et al., 2010].

Algorithm 9: Average Weighted Scheme

Local node N_i,
Require: Local centroids $C = \{c_0, c_1, \cdots, c_k\}$ and counts $\omega = \{\omega_0, \omega_1, \cdots, \omega_k\}$;
Input: Centroids and counts pairs received from neighbors set $\Gamma^{(i)}$;
Output: Average value $\bar{c} = \{\bar{c_0}, \bar{c_1}, \cdots, \bar{c_k}\}$, $|\bar{c}| = K$, where $\bar{c_k} = \frac{\bar{s_k}}{\bar{\omega_k}}$,;
Let $\{(c_j, \omega_j)\}$ be all pairs sent to node N_i;
$\bar{s_{ik}} \leftarrow \sum_j c_{jk}\omega_{jk}$;
$\bar{\omega_{ik}} \leftarrow \sum_j \omega_{jk}$;

However, this weighted average method can not be used any more when the number of clusters $\boldsymbol{k}$ of all agents in the network are not consistent. In this thesis, we consider a regeneration based method to update centroids and counts. Agents estimate the distributions of clusters based on summary description received, then combine the regenerated data with its local data and re-cluster them.

2.6.3 Convergence

When algorithm is asynchronous, which means nodes do not have to wait until they have received a message from all its neighbours, it is necessary to determinate the termination condition for nodes. For the termination conditions for nodes, there are two different ways. One method is monitoring the maximum distance between centroids of clusters, proposed by Kargupta [Datta et al., 2008]. If the maximum distance between the centroids of nodes at the start and end of iterations l is larger than a preset threshold λ, then nodes proceed to iteration $l + 1$. Otherwise it enters the termination condition. In addition, a prefixed number of maximum iterations is also used to force the node stopping updating.

$$d_i = max\left\|\omega_{j,\hat{i}} - \omega_{j,i} : 1 \leq j \leq K\right\| \tag{2.21}$$

where $\omega_{j,\hat{i}}$ and $\omega_{j,i}$ denotes the centroids for iteration $\hat{i}$ and i respectively.

The other one is global squared errors proposed by Di Fatta [Di Fatta et al.,

2013]. It firstly compute the local squared errors for agent i as follows:

$$e_i = \sum_{k=1}^{K-1} \sum_{x \in C^k} |d(\mathbf{x}, \mathbf{m}_k)|^2 \tag{2.22}$$

where $\mathbf{x}$ is dataset inside cluster C^k and $\mathbf{m}_k$ is the centroids for cluster C^k. Then compute the global errors by summing up the error from all nodes. If the global error is decreased below a specified minimum threshold λ, the agent turn to terminated condition.

2.6.4 Weaknesses of the state of the art and corresponding solutions

Distributed K-Means [Datta et al., 2008] was proposed for use in large-scale peer-to-peer wired networks. In this scheme, no raw data is exchanged, thus satisfying the basic privacy requirements, and the centroid summaries reduces the amount of data being transmitted. However, the algorithm requires synchronisation, which in wireless networks may introduce significant delay, and may fail to converge if one node has initial data whose distribution is significantly different from the average. The exchange of just centroids and counts also discards information about the shape of the cluster. When clusters have significantly different shapes, this may again cause problems for convergence.

Bendechache et al. [Bendechache et al., 2016] represent each cluster by its boundary points, and then exchange the boundary and the number of internal points. Messages flow up the tree to the root. This approach solves the issue of arbitrary shapes, but introduces failure points at each aggregation, and requires a secondary communication from root to leaves to disseminate the final clusters. It also exposes some of the raw data to neighbouring nodes, in order to specify the cluster boundary.

Gossip-based methods for distributed K-means are proposed in [Di Fatta et al., 2013, Bénézit et al., 2010]. Although global synchronisation is avoided, local synchronisation is still required. In practice, the nature of the gossip algorithm means that many rounds are required before convergence. Since there is no global synchronisation, nodes are unaware of when other nodes have converged, and so another algorithm running in parallel is required to detect termination, or the algorithm is forced to terminate after a fixed number of rounds.

Due to communication cost, privacy consideration and power constraints of wire-

less network, in-network learning methods should be efficient in energy consumption and aware of latency. For the high computation cost, distributed dictionary learning technology may not be suitable. However, distributed clustering algorithm is a promising solution, but with the following weaknesses:

1. Only observations from the same source is considered. However, there might be sub-groups that observed different scenes or readings in some practical applications and this must be identified. For instance, agents that are close to a fire may have a much higher temperature reading that other agents and this must be reported immediately. Based on these concerns, an in-network method that can distinguish different data patterns across the network is desired.

2. The number of clusters is preset. As mentioned before, K-means requires the number of clusters $\boldsymbol{k}$ to be fixed in advance. It is a strong assumption for unsupervised learning, and if the methods are automated and act in response to observed data, then it is unrealistic to assume that the $\boldsymbol{k}$ value will be known. In addition, unless we can guarantee that all nodes are observing the same phenomenon, then it is possible that the appropriate $\boldsymbol{k}$ value will vary across the nodes. But if we allow different nodes to use different $\boldsymbol{k}$ according to their own data, then we have a problem with limited data fractions - a node may be misled by its observed data and propose a smaller $\boldsymbol{k}$ value than is appropriate for the true data. Another concern is that the overlapping between clusters. Huge overlapping between clusters will force number of clusters $\boldsymbol{k}$ deviate from the true value. Based on these consideration, an adaptive distributed K-means clustering for in-network learning is desired.

3. The weighted average aggregation function only be suitable for the case where the number of clusters $\boldsymbol{k}$ is fixed to be same for all nodes. So a data aggregation function with adaptive number of clusters is desired.

4. Each agent has to synchronise its behaviour with its neighbours [Bendechache et al., 2016, Datta et al., 2008, Di Fatta et al., 2013, Kotary and Nandal, 2019, Bénézit et al., 2010]. Synchronisation in wireless networks may introduce significant delay.

5. Communication cost is ignored. Since the power of node is limited and in-network communication consumes the largest part of energy, it is necessary to reduce the transmission of message to prolong the lifetime of network

[Miller and Vaidya, 2005].

Before we describe the work in this thesis, we introduce the methods to measure the performance of clustering:

1. Accuracy against ground truth data points. When we know the ground truth distributions to create the raw data points, we compare the clustering results with the ground truth data points. Any two data points that are assigned into the same cluster or assigned into different clusters as ground truth distributions are counted.

2. Accuracy against centralised data points. When the ground truth distributions are not available, we run a single centralised clustering to get a benchmark clustering result. Then compare our clustering result with this benchmark result.

3. Accuracy against patterns. In cases of multiple patterns, where there are sub-groups of agents receive data from different resource, we measure how accurately the algorithm put agents into the same group as ground truth patterns.

In this thesis, first, we will propose an asynchronous distributed clustering algorithm for known $\boldsymbol{k}$ values that can also handle cases where the number of nodes in the network is not known. The algorithm is able to reproduce clusters that were used to create the raw data, it avoids transmitting raw data across the network, and it converges in reasonable time. It is based on the idea of sharing information about the data distribution in each cluster, and it can be used with different internal cluster algorithms. Then we develop a revised algorithm that does not require the $\boldsymbol{k}$ value as input, and we show that we can retain accuracy and the good time performance. Then we introduce a method which can handle different sub-groups of nodes observing different phenomena, with possibly different number of clusters. The method is based on applying a similarity measure based on an adapted EMD to the different cluster models, but it does assume there is more than one pattern to be detected. Finally, we introduce a method that can can identify whether or not there are multiple patterns, based on the idea of anomaly detection.

Chapter 3

Asynchronous Distributed Clustering for Mesh Networks

We consider a problem motivated by Wireless Mesh Networks: how should the agents act and communicate so that all agents end up with the same understanding of the environment? We assume this is an unsupervised task, and so we model the problem as clustering. For this chapter, we assume that all agents are sensing the same phenomenon, and so should finish with the same cluster descriptions, and we assume that the number of clusters to be described is known by all agents at the start. Our aim is to enable this while minimising elapsed time and communication costs, without imposing a central controller in the network. For both privacy and efficiency, we want to avoid sharing raw sensed data values; instead, we will share summary descriptions.

Each of the different algorithm variants is constructed from individual runs of a clustering algorithm by each agent, followed by sharing of cluster descriptions across the network. The description includes information on the size and 'shape' of the clusters. Depending on the particular scenario, after receiving new information from other agents, an agent may update its own cluster description before sharing it again, or it may simply relay the received information. When updating the cluster descriptions, in order to exploit well-known clustering algorithms, an agent may regenerate sample data points from the received information. Once an agent believes it has received information from enough agents, it computes a global cluster description, and shares that.

The outline of this chapter is as follows: In the first section 3.1, the main algorithm is described in detail. Then in section 3.2 we introduce a regeneration based

method to describe the shape of clusters. After that, two different scenarios are illustrated in section 3.3 and experimental result are shown in section 3.4. Conclusion is given in section 3.6.

3.1 Main Framework

In this section, we describe the asynchronous distributed clustering algorithm (ADC) for all agents to learn wider network pattern (for instance, understand the environment) (algorithm 10 and 11). We are assuming a flat ad hoc network where each node has a limited communication range (for example in wireless networking). Therefore each node has a set of neighbours with which it can communicate directly. We do not assume any distinguished node in the network, and so any node may be first to initiate the clustering process. Our proposed algorithms do not impose any topology on top of this network, although we do compare to algorithms for which a communication tree has been built, to get a baseline for our results.

Algorithm 10 shows the outline algorithm when agents know the size of network. Different instantiations of this framework will be developed in this and following chapters. The node clusters its local data with a pre-defined $\boldsymbol{k}$. It then summarises the result of the clustering, and transmits that summary to neighbours. After receipt of cluster summaries from each agent, the agent processes those descriptions and do local clustering. Once a node detects that successive iterations do not change the clusters significantly, and that there are no incoming messages, it terminates.

Figure 11 shows the basic step to compute the global model when the size of network is not known. Compared to the previous scenario, agent has to compute the provisional model whenever received new massage. The agent will turn to sleep but is ready to wake up. The algorithm will terminate if have not received any message for a specified seconds.

3.2 Cluster descriptions and integration

Transmitting cluster centroids and counts is not enough, since it loses important information about the distribution of data points around the centroids. We propose that each cluster is described by centre, estimation of size, and a description of the distribution and shape. In this section, we present three different schemes

Algorithm 10: Asynchronous distributed clustering algorithm (a)

```
/* The size of network is known                                    */
Initialization: generate initial cluster model with a pre-defined k;
Exchange cluster descriptions with neighbours;
while not stabilised do
    if received messages then
        process received summaries and transmit results;
        if received all summaries then
            Compute the global model;
            Send convergence command to neighbours ;
            Set stabilised to be true;
        else if Receive terminate message (a.k.a., global model) then
            Update local knowledge;
            Transmit terminate message to all neighbours;
            Set stabilised to be true;
    else
        Wait until receive a message;
        if have not received any message for a specified seconds then
            Set stabilised to be true ;
```

Algorithm 11: Asynchronous distributed clustering algorithm (b)

```
/* the size of network is not known                                */
Initialization: generate initial cluster model with a pre-defined k;
Exchange cluster descriptions with neighbours;
Wait until messages received or time exceeded ;
while time not exceeded and messages received do
    Incorporate received neighbours' cluster information;
    Run local clustering to completion;
    Transmit new summaries to neighbours;
    Wait until message received or time exceeded;
```

for this description. Figure 3.1 shows the three different schemes:

1. K-means. Local data are clustered by K-means. Nested bounding boxes are then used to describe the shape of clusters. Centroids, counts and bounding boxes are shared with neighbours.

2. Separate Gaussians (SG). Similarly, local data are clustered by K-means and then a separate multi-dimensional Gaussian is fitted to each cluster, where the shape of cluster is denoted by mean and variance. In this case, the Gaussians, plus a count of the data points in each cluster, are shared with neighbours.

3. Gaussian Mixture Model (GMM). A Gaussian Mixture Model is fitted to the local data by the EM algorithm. The shape of each cluster is described by the means and variances of the GMM. The GMM, plus a count of the number of data points in each cluster, is shared with neighbours.

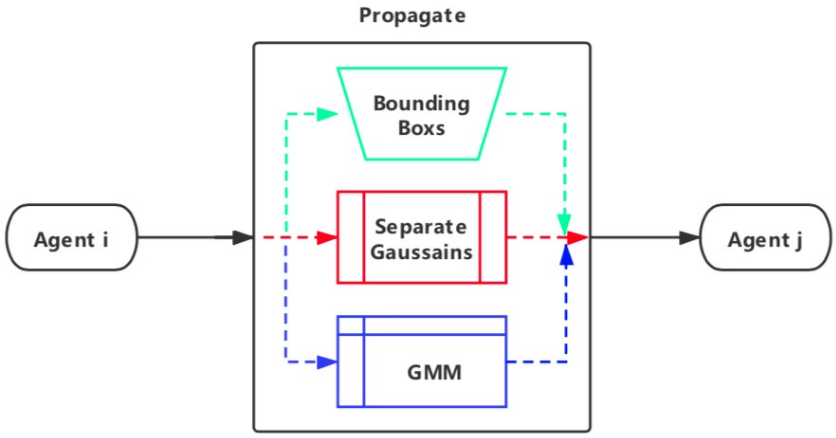


Figure 3.1: Information propagation

The standard output from K-means is a list of centroids, with no associated information on the shape of the identified cluster, whereas the cluster descriptions from SG and GMM do include shape descriptors – the mean and variance (or co-variance matrix). Therefore, we augment the output of K-means by deriving nested bounding boxes. We describe them using centroids, size of clusters principal components, and percentiles with their corresponding values (for instance, percentile 5% and value 0.1 means that 5% of data points is less than 0.1).

Suppose the percentile used for each axis is [0.0, 0.1, 0.3, 0.7, 0.9, 1.0]. We first apply Principal Component Analysis to each cluster, to generate the axes of the cluster shape, centred on the centroid. For each positive and negative axis, we generate the maximal data point, the 80th percentile point, and the 40th percentile point. We then adjust the 80th and 40th percentile points to midway between that point and next closest greater point. Taken together, this produces three bounding boxes (or hyper-rectangles), containing 100%, 80% and 40% of the points respectively, oriented along the PCA axes, and thus approximates the shape and density of the points in the cluster. Please note that this method assume that data points inside a cluster are approximately evenly scattered.

Figure 3.2 shows on example how to describe the summary description given a raw dataset. The red bounding box shows all raw data points are located inside this box, the two arrowed direction shows the main orientation of shape of this

cluster, and the cross denotes the centroids of this cluster. Note that PCA is only carried out on the whole cluster, which indicates that the axes are the same for each box. The computation cost is the main reason why we do not do PCA for each small percentile box.

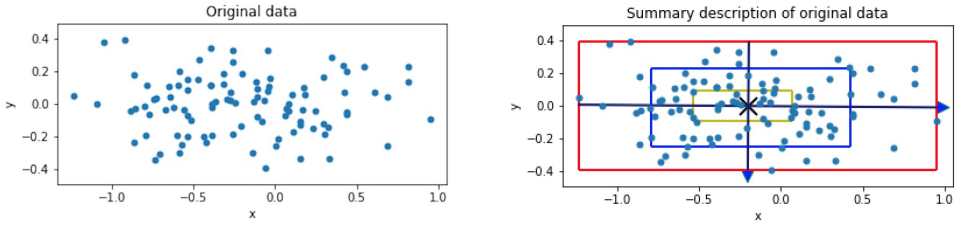


Figure 3.2: Original data and its summary description

At some stages in the overall algorithm, an agent will need to combine cluster models from two or more agents. Possible methods include merging of the descriptors by generating a new set of bounding box that enclose the individual ones, or mathematical models for combining Gaussians. Instead, we consider methods which generate new sets of data points by sampling from the distributions or descriptions, and then re-running local clustering. If receives a GMM or separate Gaussians, agents uses standard techniques to sample from these distributions, proportional to the associated counts.

When the received description is a bounding box model, we need to specify how to sample data points, and the process is shown in Algorithm 12. First, randomly generate an x coordinate and find its right place by comparing with the percentiles, then generate a random number in the corresponding interval of percentile values. The number of points n to generate in a particular area can be inferred by the bounding boxes. For example, assume the array of box percentile values is $[0.0, 0.1, 0.3, 0.7, 0.9, 1.0]$ and the corresponding values are $[v0, v1, v2, v3, v4, v5]$. If $A = 0.5$ is generated, since 0.5 is located in the third interval $[0.3 - 0.7]$ of the percentile, the value of A should be randomly generated from the range $[v2, v3]$). Similarly, generate a y coordinate and store the coordinate pair as a new point.

Figure 3.3 shows a summary description, and the data points generated from it. The evaluation of this regeneration based method is attached in the appendix 8.1, where we consider how different percentiles for the bounding boxes, different number of data points and different underlying distributions affect the performance. Experiments show that we could generate a similar cluster based on its summary description generated by our method. Descriptions for clusters that are

Algorithm 12: Generate new data points

Input : Number of points to generated n, array of box percentile p and corresponding values s_{val}
Output: Generated dataset n_{da}

count = 0 ;
while *count* $< n$ **do**
    /* generate x coordinate */
    Random generate a number A in $[0, 1]$;
    Find the interval i of percentiles in p that contains A ;
    Generate a random number in the corresponding interval of percentile values $s_{val}[x]$: x_{tem} = sample from uniform($s_{val}[x][i]$, $s_{val}[x][i+1]$) ;
    /* generate y coordinate */
    Random generate a number B in $[0, 1]$;
    Find the interval j of percentiles in p that contains B ;
    Generate a random number in the corresponding interval of percentile values $s_{val}[y]$: y_{tem} = sample from uniform($s_{val[y]}[j]$, $s_{val[y]}[j+1]$) ;
    n_{da}.append($[x_{tem}$, $y_{tem}]$);
    count += 1 ;

Algorithm 13: Regeneration algorithm

Input: Summary descriptions $S = \{S_1, S_2, \cdots, S_k\}$;
Output: Regenerated dataset R_s;
for *each cluster description* $S_i \in S$ **do**
    if *Clustering algorithm is K-means* **then**
        /* p: array of box percentile, s_{val}: corresponding values of p */
        $S_i = \{n, p, s_{val}\}$;
        Call algorithm 12;
        Append generated dataset to R_s*;*
    else if *Clustering algorithm is GMM or SG* **then**
        /* μ: mean of cluster, *val*: the variance of cluster */
        $S_i = \{n, \mu, var\}$*;*
        Randomly generate n data points based on μ *and val;*
        Append generated dataset to R_s*;*

elliptical are more stable than those for circular clusters. One possible approach to improve the quality of generated cluster is choosing an appropriate percentile.

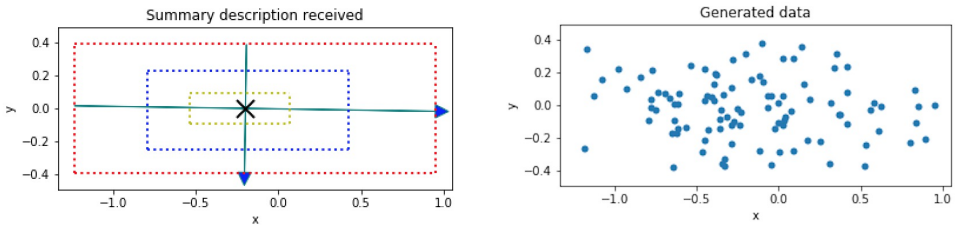


Figure 3.3: Bounding boxes received and regenerated data based on it

3.3 Distributed Clustering algorithm and various scenarios

In the last section, we have shown that, given a cluster of data points, we can summarise it and then generate new points that give approximately the same description. Now the question is how to share these summary descriptions and update local models when a new model is received.

While a node is doing local clustering, more messages may arrive from neighbours. We considered two approaches: transmit the result of local clustering, and then read any stored messages and repeat; read stored messages and repeat clustering, until the inbox is empty, and then transmit the clustering results. In practice, transmitting before reading new messages produced better performance, so we only report those results. As well as transmitting their own summaries, we allow each node to send on summaries received from other nodes without modification if needed. Further, to each summary we attach a list of all nodes whose data was used to generate the clusters. This supports the delayed clustering above, where a node waits until it has received information from all known nodes. In some cases, we also allow a node to issue a request for reduced data summaries (summaries built from a specific subset of nodes). Finally, we consider case where messages are sent to all direct neighbours, and where messages are sent to a randomly selected subset of neighbours.

For incorporating the simplest cluster summaries, we compute the weighted average centroids. For cases where we receive more descriptive summaries, we generate new data points by sampling from those descriptions, and we sample in proportion to the cluster counts. The sampled data is then combined with the local data, and provides the input to next round of local clustering. For the PCA/bounding box description, we generate points uniformly at random in the 40% box, and then generate points uniformly in each sector to extend to the

80% box, and then repeat to fill the remainder of the 100% box (Fig 3.3, for the same case as Fig 3.2). For the Gaussian summaries, we simply sample from the Gaussian distribution.

We can use our framework to recreate the method of [Datta et al., 2008] (use 'no further improvement' as terminating condition), by all nodes starting with the same pre-scheduled initial centroids, running a single internal iteration of k-means, counting the data points assigned to each cluster, transmitting a list of pairs of centroid location and centroid count to all neighbours, waiting until all neighbours have transmitted, then computing a weighted average location for each centroid. Similarly, we can replicate the method of [Di Fatta et al., 2013], by all nodes starting with the same pre-scheduled initial centroids, running a full iteration of k-means, counting the data points assigned to each cluster, transmitting a list of pairs of centroid location and centroid count to a single randomly selected neighbours, then computing a weighted average location for each centroid.

3.3.1 Agent knowledge: known number of nodes

We consider two different scenarios for the initial knowledge nodes have of the rest of the network. In this first scenario, we assume that each node in the network knows the total number of other nodes. In any uncontrolled process, we run the risk of creating a feedback problem, where the neighbours of a node send it summaries which already incorporate that node's own cluster descriptions. For example, A sends its summaries to B, which incorporates the data, generates new summaries, and sends them on to C; C updates its global view, and passes it on to A; if A then incorporates that model into it local data, it will effectively give double the weight to its own data.

To avoid this problem, each agent can simply transmit its own summaries labelled with its own ID, and then simply relay other received previously unseen summaries without re-clustering. Once it has received the correct number of summaries, it can combine them all to get a global picture. At that point, it can transmit the global summary, with a flag to indicate that no new information will be received. Each agent receiving this termination message stops its own process, adopts the global summary, and retransmits the message. We call this algorithm variant Model Merge after Filtering 1 (MMF1), shown in Algorithm 14. We note that there is some privacy loss here, since at least one agent will see the individual summaries from each other node.

Algorithm 14: MMF1 for node N_i in n node network

```
Require: Message box B;
Input  : Local dataset X_i
Output: Final representatives of clusters
Randomly generate k initial centroids in the space;
Cluster X_i with the generated centroids as input;;
Initialize list of received model T;
while not terminated do
    if Message box B is not empty then
        if Receive terminate message then
            Send terminate message to all neighbours;
            Terminated
        else if Receive data message x and x ∉ T then
            Add x to T;
            if len(T) == n then
                Call algorithm 13;
                Combined regenerated data with local data;
                Do local clustering on the combined data;
                Compute global view;
                Send converge command to neighbours;
            else
                Share message x with neighbours;
```

3.3.2 Agent knowledge: direct neighbourhood only

In this scenario, we assume that each node knows only its immediate neighbours, and does not know the identity of other nodes, or even the size of the network. The same feedback problem as in the previous scenario still applies. In this case, we cannot wait until all summaries are received, since no node knows how many summaries are expected. Instead, we exploit the IDs that we can attach to each model.

When a node receives a summary constructed only from IDs it has not seen before, it applies the basic summary incorporation approach. If it receives a summary constructed entirely from IDs it has already seen, it ignores it. If it receives a summary built from some old and some new IDs, it then applies a model subtraction procedure, to avoid the feedback problem. It generates sample data points for the parent model, then generates temporary data points for the model to be subtracted, and for each temporary data point, it removes the closest data point generated for the parent model. If it does not have an appropriate set of smaller models to subtract, it identifies a small missing set, and sends a request

to one or more neighbours which transmitted a superset. Those neighbours either reply with the summaries, or issue their own requests in turn. The algorithm, MMF2, is described in Algorithm 15.

Algorithm 15: MMF2 for node N_i

Require: Message box $\boldsymbol{B}$;
Input : Local dataset X_i
Output: Final representatives of clusters
Randomly generate k initial centroids in the space;
Cluster X_i with the generated centroids as input;
Send representatives and model name M_i to neighbours;
Wait until messages received or time exceeded ;
while *time not exceeded and messages received* **do**
    Incorporate received neighbours' cluster information;
    Call algorithm 16;
    Run local clustering to completion;
    Transmit new summaries to neighbours;
    Wait until message received or time exceeded;

Algorithm 16 shows the procedure for model subtraction. First, the final model $\boldsymbol{f_{model}}$, which contains all unique messages it saw, is computed and the largest subset $\boldsymbol{l_{sub}}$ of $\boldsymbol{f_{model}}$ is computed as well. For instance, sensor i received two messages: abc and $bcef$, then the final model $\boldsymbol{f_{model}}$ is $abcef$ and $\boldsymbol{l_{sub}}$ is $bcef$.

After that, $\boldsymbol{d_{set}}$ is defined as the difference between $\boldsymbol{l_{sub}}$ and $\boldsymbol{f_{model}}$. Then, the elements of $\boldsymbol{d_{set}}$ are searched for in History table $\boldsymbol{T}$. If any item is still missing, the agent sends out a request to specific neighbours who have transmitted this message before. Finally, models for $\boldsymbol{f_{model}}$ are combined by substracting and regenerating and the regenerated dataset is clustered to get the new model.

3.3.3 Time Complexity

Algorithm 14 and 15 consists of updating local model, generating data points and computing the global models. The time complexity to generate N data points is bounded by $\mathcal{O}(N)$. If the size is known in advance, an agent computes the global model as soon as it receives local model descriptions for each node in the network. In this case, the time complexity depends on the clustering algorithm used to compute the local and global models.

1. K-means is used as clustering algorithm. Since we calculate the distance in a 2-dimensional space, we assume the cost of computing the distance

Algorithm 16: Model Subtraction

Input : History table $\bar{\boldsymbol{T}}$, Message list $\boldsymbol{m_i}$ $(0 \leq i \leq N)$
Output: regenerated dataset R_s

```
H ← set of node IDs in T̄;
f_model ← union of H & node IDs in m_i;
T̄ = T̄ + m_i;
if there exists a set of model IDs S ∈ T̄ such that S = H then
    f_model = S
else
    l_sub ← biggest subset from T̄;
    d_set ← f_model \ l_sub;
    Find elements of d_set missing from T̄;
    Send request to neighbours to get those models;
    f_model = l_sub + d_set;
Call algorithm 13 with f_model as input;
Cluster regenerated dataset;
Summarise the clusters;
return new model of clusters;
```

from a point to a cluster centroid is constant. Running t iterations of K-means loops (Lloyd's algorithm) exhibits the complexity of $\mathcal{O}(N\boldsymbol{k}pt)$, where $\boldsymbol{k}$ is the number of clusters, p is the dimensionality of the data [Bachem et al., 2016]. Usually the iterations t and $\boldsymbol{k}$ are small, so the time complexity is $\mathcal{O}(N)$ [1] [Arthur and Vassilvitskii, 2006]. In this case, PCA is used to compute the nested bounding boxes. Computing PCA involves a time complexity of $\mathcal{O}(p^2 * h + p^2 * N)$ [Sharma and Paliwal, 2007], where p is the dimension of the data and h is the number of eigenvector in a reduced dimension to be computed [2]. Given the underlying dataset is 2-dimensional, computing the PCA takes time in $\mathcal{O}(N)$. So the overall time complexity is bounded by $\mathcal{O}(N + N + N + N) = \mathcal{O}(N)$.

2. GMM is used as clustering algorithm. The time complexity of GMM is $\mathcal{O}(N * \boldsymbol{k} * p^3) = \mathcal{O}(N)$ [Pinto and Engel, 2015]. The overall time complexity is bounded by $\mathcal{O}(N + N + N) = \mathcal{O}(N)$.

3. SG is used as clustering algorithm. Compared the process with that of K-means, the computation of PCA is not required. Applying a Gaussian process to a dataset of size N has complexity $\mathcal{O}(N^3)$, and it could be reduced

[1] In the worst case, k-means is exponential.
[2] When p is larger than N, the time complexity is bounded by $\mathcal{O}(min(p^3, N^3))$ [Johnstone and Lu, 2009]

to $\mathcal{O}(N)$ by various approximate techniques (i.e., partition the dataset into several groups) [Hensman et al., 2013]. Since we fit each cluster to a separate Gaussian, so the overall complexity is $\mathcal{O}(\boldsymbol{k}\bar{N})$, where $\bar{N}$ is the size of cluster and $\bar{N} < N$. So the overall time complexity is bounded by $\mathcal{O}(N + \boldsymbol{k}\bar{N} + N)$ $= \mathcal{O}(N)$.

When agent is only aware of its immediate neighbours, agent has to compute the provisional global model whenever received new message. Suppose an agent has to update T rounds until convergence. The complexity of algorithm varies with the clustering algorithm:

1. K-means is used as clustering algorithm. In this case, an agent has to compute PCA, re-generate data points and do clustering for T rounds. So the overall time complexity is bounded by $\mathcal{O}(N + T(2N + N)) = \mathcal{O}(TN)$.
2. GMM is used as clustering algorithm. The overall time complexity is bounded by $\mathcal{O}(N + 2TN) = \mathcal{O}(TN)$.
3. SG is used as clustering algorithm. The overall time complexity is bounded by $\mathcal{O}(N + 2TN)) = \mathcal{O}(TN)$.

3.4 Empirical Evaluation

In this section, we evaluate the performance of proposed methods and compare it to the state of the arts. First, we make some assumptions about the underlying network. The network

1. is connected – there is a multi-hop path between any pair of nodes;
2. uses point-to-point communication – i.e. each message transmission is sent to a single neighbouring node;
3. ensures reliable delivery of the messages, but this may require retransmissions on failure, and so there is a message-specific delay for any transmission;
4. begins clustering at a pre-scheduled time – all nodes start clustering at approximately the same time, with the same expected number of clusters;
5. supports a parallel process to detect termination of the algorithm.

For the environmental conditions, we assume that nodes gather their own data (i.e., from sensors or human interaction), and that they have a data storage policy which determines the length of time for which that data is active. We assume that

this distributed clustering task is then taking a snapshot of the current local data stores. We make no particular assumptions about the form of the data, apart from assuming we can apply various well-known algorithms to do local clustering. In practice, in the thesis, we restrict our data samples to 2-dimensional data.

We evaluate a number of different instantiations of the ADC framework. MMFi-P uses full k-means, exchanging PCA and Bounding Box summaries, regenerating data by sampling, for the two MMF variants. Similarly, MMFi-SG uses full k-means, but then exchanges separate Gaussian models with counts fitted to each cluster. MMFi-GMM fits a Gaussian Mixture Model for clustering, and exchanges the GMM with counts. We also evaluate [Datta et al., 2008] partial k-means, exchanging centroids and counts with all neighbours, and [Di Fatta et al., 2013, Bénézit et al., 2010] (full k-means, exchanging weighted centroids and counts with a single neighbour).

To simulate the operation of the algorithms on a mesh network, we implement an Asynchronous Message Delay Simulator, based on [Zivan and Meisels, 2006]. Random network topologies are generated based on [Erdös and Rényi, 1959], where the probability p_N is larger than $\frac{(1+\varepsilon)ln(N)}{N}$, where ε is a positive constant, to ensure that a random graph generated is a connected graph (we set $\varepsilon = 1$). To generate dense graph topologies, p_N is set to 0.8.

We consider a wireless network (dense or sparse) with 10 agents. The initial raw data is generated in $\boldsymbol{k}$ clusters, sampled from a two-dimensional symmetric Gaussians, scaled to the range [0,1] before being assigned randomly to network nodes. Each node is given 200 data points and the number of clusters in all simulations is set to 5. Each message transmission delay is generated uniformly from the range [0.5, 1.0], and nodes begin their work at time randomly selected in [0, 0.1s].

Before we distribute the data points over the nodes, we run a single centralised k-means, to get a benchmark clustering result. Agents' clustering results are then compared to the clustering of centralised method. We use the percentage of membership mismatch (PMM) as our formal measure of clustering error:

$$PMM^{(i)} = 100\frac{|\vec{x} \in X^{(i)} : L_c(\vec{x}) \neq L_p^i(\vec{x})|}{|X^{(i)}|} \tag{3.1}$$

where $L_c(\vec{x})$ denotes the label of the cluster to which $\vec{x}$ is assigned at the end of centralized methods and $L_p^i(\vec{x})$ denotes the label of the cluster to which $\vec{x}$ is assigned once the node reaches the termination state.

Table 3.1: Notation

MMF1	MMF scheme under Scenario 1
MMF2	MMF scheme under Scenario 2
P	Describe clusters by bounding boxes
SG	Fit clusters by separate gaussians
GMM	Gaussian Mixture Model
R-msg	Received data message
R-poll	Received poll message
t_0	First agent learned global view
t_1	Last agent learned global view
t_2	Convergence time for algorithm

Since GMM is a soft clustering method, which assigns to a single data point a probability of membership of each cluster, PMM may be unfair to GMM, since it can never achieve 100% clustering accuracy. In order to compare with the hard assignment obtained by k-means, we assume that the final assignment of a data point to a cluster is determined by the highest probability. In addition to clustering accuracy, we measure the convergence time, which records the interval between the time when first agent initialised and last agent terminated. For communication costs, we record the total number of received messages. For each measure, we compute Mean and Standard Deviation over 50 random problem instances at each setting. The notation is summarised in Table 3.1.

None of our methods use a central controller, and we assume an additional process which detects termination if required. For termination time, we record the last time at which any node or message was active. For MMF2, we also measured the time that the first node reached stability (t_0), and the time that the last node reached stability (t_1). In those MMF2 experiments, t_2 is the final termination.

The number of messages received is not necessarily sufficient to measure communication cost, since large messages require more packets and thus longer transmission times. For IEEE 802.11ac, the maximum size of a single packet (MPDU) is 11,454 bytes. For our experiments, we limit the number of clusters to 5. Five centroid locations plus counts, or five Gaussians plus counts, can be described in just over 1000 bytes. The principal components and bounding boxes for five clusters can be described in just 9,272 bytes, and thus, for our experiments, each model description can be transmitted in a single packet, and so the total number of messages is a good proxy for communication energy costs.

First we evaluate performance under scenario 1, in which nodes know the size of the network. The results for densely connected networks of 10 nodes are shown

in figure 3.4. MMF1-P and -SG show the highest accuracy (relative to centralised k-means); Both of these methods use full k-means at each node to generate the initial clustering and the final clustering, but differ in the summaries that are exchanged. MMF1-P requires the fewest messages, but takes significantly longer than MMF1-SG to terminate – this appears to be because of the extra time required to analyse the clusters using PCA and to generate the bounding boxes. MMF1-GMM requires the least time to stabilise, and achieves reasonably high accuracy. The existing methods from [Datta et al., 2008], [Di Fatta et al., 2013] and [Bénézit et al., 2010] are outperformed by the three MMF1 variants on all measures, confirmed by a Tukey Test with significance level p=0.001. In Figure 3.5, we show the result for sparsely connected networks of 10 nodes. MMF-1 achieves the highest accuracy and requires the least amount of messages to converge. Again, MMF1-GMM converges faster than other methods and achieves high accuracy. Again, [Datta et al., 2008], [Di Fatta et al., 2013] and [Bénézit et al., 2010] are outperformed on all measures (Tukey Test, p =0.001). To summarise, selecting between MMF1-P, -SG and -GMM will depend on the relative costs of accuracy, message transmission, and elapsed time.

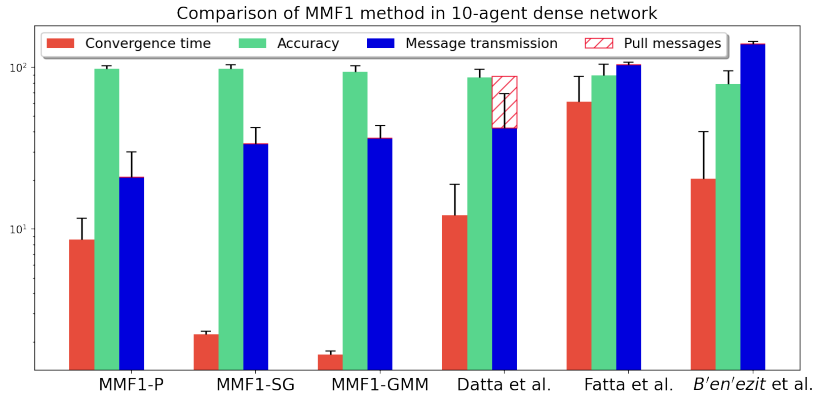


Figure 3.4: Comparison of MMF1 method in 10-agent dense network, showing mean results for each measure over 50 runs.

In figure 3.6 we show the result of for densely connected networks of 10 nodes for scenario 2, in which nodes only know of the existence of their immediate neighbours. We report three different time measures. MMF2-P again requires fewest messages. MMF-SG again has the highest accuracy. The relative performance of MMF2-GMM has improved, with close to the highest accuracy, and clearly the fastest termination time. [Datta et al., 2008] is still outperformed by the other methods, although the number of messages (including both data messages and polling messages) are now closer in value. [Di Fatta et al., 2013] and [Bénézit

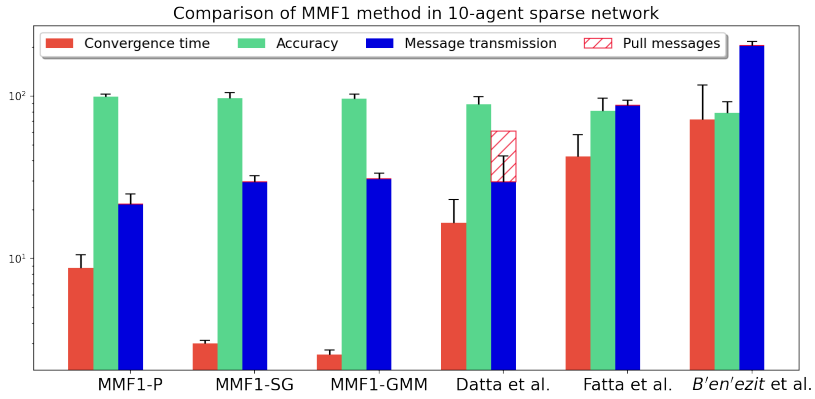


Figure 3.5: Comparison of different schemes under MMF1 method in a 10-agent sparse network, showing mean results for each measure over 50 runs.

et al., 2010] are still outperformed on all measures. The result for sparse networks are shown in figure 3.7. MMF2-SG shows the highest accuracy and MMF2-P requires the fewest messages but takes substantially longer time than MMF2-SG to terminate. MMF2-GMM converges much faster than other methods and still achieves high accuracy. The methods in [Datta et al., 2008], [Di Fatta et al., 2013] and [Bénézit et al., 2010] are outperformed on all measures (Tukey Test, p = 0.001).

In summary, the methods that use full clustering at each node on each cycle, and which exchange more informative descriptions, outperform [Datta et al., 2008], [Di Fatta et al., 2013] and [Bénézit et al., 2010]. Taking MMF2-GMM as representative, it achieves 10 percentage points higher accuracy relative to centralised k-means (reduces the misclassification rate from 12% to 1.7%), reduces the message count by 20%, and reduces elapsed time by 75%.

In all cases, we respect the privacy of the original data, and do not exchange any individually identifiable data points. However, we do lose some privacy compared to [Datta et al., 2008], in that we can now identify data distributions for individual nodes.

3.5 Significance Study

In this section, we discuss the statistical significance of the results. Most of the parametric tests require that the assumption of normality be met. Usually, a Shapiro-Wilk's W test is used to test the assumption of normality, then a parametric test is carried out. However, if the sample sizes are large enough

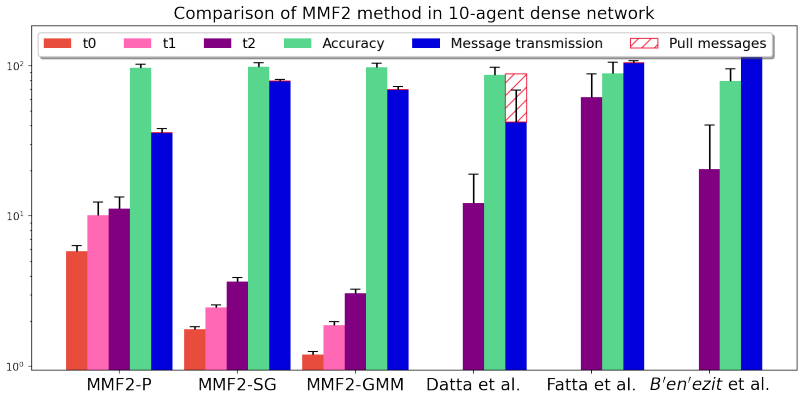


Figure 3.6: MMF2 method in 10-agent dense networks

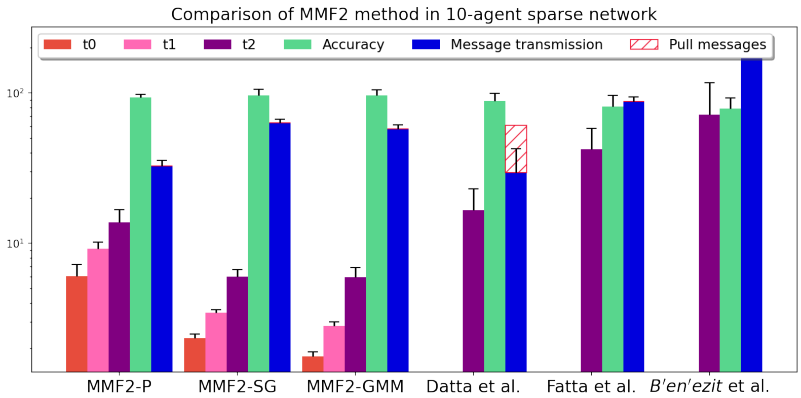


Figure 3.7: Comparison of different schemes under MMF2 method in a 10-agent sparse network

($\zeta > 30$), the violation of the normality assumption should not cause major problems and we can use parametric procedures even when the data are not normally distributed [Ghasemi and Zahediasl, 2012].

A statistical comparison is thought to be significant when its P-value is less than 0.05. We apply the Tukey test to conduct pairwise comparisons of the approaches. The Tukey test is commonly thought to be better than the t-test, as it takes the Type I error (false positives) into account [Lee and Lee, 2018]. When doing multiple comparisons, Tukey test considers the entire collections of pairwise hypothesis tests. With the significance level of 5%, it assures that there is at most a 5% chance to obtain a false positive among the entire set of pairwise tests. This is very different from simply doing pairwise tests individually with significance level of 5%.

The statistical analysis is carried out through the built-in functions in Python. Table 3.2 shows that, for all the scenarios, the null hypothesis is rejected with the P-values all less than 0.05. It indicates that there is significant difference between proposed algorithms and the state-of-the-art and the improvement is significant.

Table 3.2: Pairwise Tukey test for proposed methods and the state-of-the-art. $\diamond$ denotes the difference of mean.

	Dense Graph			**Spare Graph**		
MMF1	Time ($\diamond$, P value)	Accuracy ($\diamond$, P value)	R-msg ($\diamond$, P value)	Time ($\diamond$, P value)	Accuracy ($\diamond$, P value)	R-msg ($\diamond$, P value)
$P-Datta$	(-3.6, 0.001)	(11.30, 0.001)	(-67.80, 0.001)	(-8.0, 0.001)	(10.20, 0.001)	(-38.50, 0.001)
$P-Fatta$	(-53.0, 0.001)	(9.20, 0.001)	(-84.0, 0.001)	(-33.6, 0.001)	(17.5, 0.001)	(-66.0, 0.001)
$P-Bénézit$	(-12.0, 0.001)	(20.1, 0.001)	(-119.67, 0.001)	(-63.0, 0.001)	(20.4, 0.001)	(-183.1, 0.001)
$SG-Datta$	(-10.0, 0.001)	(11.5, 0.001)	(-54.0, 0.001)	(-13.6, 0.001)	(8.40, 0.001)	(-31.0, 0.001)
$SG-Fatta$	(-59.0, 0.001)	(9.20, 0.001)	(-71.0, 0.001)	(-39.3, 0.001)	(15.70, 0.001)	(58.20, 0.001)
$SG-Bénézit$	(-18.3, 0.001)	(16.3, 0.001)	(-18.56, 0.001)	(-68.30, 0.001)	(18.60, 0.001)	(-185.2, 0.001)
$GMM-Datta$	(-10.6, 0.001)	(7.40, 0.001)	(-19.0,0.001)	(-14.0, 0.001)	(7.70, 0.001)	(-30.0, 0.001)
$GMM-Fatta$	(-60.6, 0.001)	(5.20, 0.001)	(-68.1, 0.001)	(-39.60, 0.001)	(15.0, 0.001)	(-67.3, 0.001)
$GMM-Bénézit$	(-19.0, 0.001)	(15.30, 0.001)	(-103.0, 0.001)	(-69.03, 0.001)	(17.8, 0.001)	(-174.1, 0.001)
	Dense Graph			**Spare Graph**		
MMF2	Time ($\diamond$, P value)	Accuracy ($\diamond$, P value)	R-msg ($\diamond$, P value)	Time ($\diamond$, P value)	Accuracy ($\diamond$, P value)	R-msg ($\diamond$, P value)
$P-Datta$	(-1.13, 0.001)	(9.7, 0.001)	(-52.0,0.001)	(-3.16, 0.001)	(4.50, 0.001)	(-27.3, 0.001)
$P-Fatta$	(-50.50, 0.001)	(7.6, 0.001)	(-68.30, 0.001)	(-29.40, 0.001)	(11.8, 0.001)	(-137.5, 0.001)
$P-Bénézit$	(-8.8, 0.001)	(18.9, 0.001)	(-104.70, 0.001)	(-58.10, 0.001)	(14.60, 0.001)	(-172.54, 0.001)
$SG-Datta$	(-9.30, 0.001)	(11.19, 0.001)	(-8.20,0.001)	(-10.7, 0.001)	(8.02, 0.001)	(3.30, 0.001)
$SG-Fatta$	(-58.0, 0.001)	(9.1, 0.001)	(-45.20, 0.001)	(36.6, 0.001)	(15.36, 0.001)	(-24.20, 0.001)
$SG-Bénézit$	(-17.0, 0.001)	(20.30, 0.001)	(-60.0, 0.001)	(-65.70, 0.001)	(8.02, 0.001)	(-141.10, 0.001)
$GMM-Datta$	(-8.90, 0.001)	(20.70, 0.001)	(-18.30,0.001)	(-10.70, 0.001)	(7.60, 0.001)	(-2.20, 0.001)
$GMM-Fatta$	(-58.30, 0.001)	(8.60, 0.001)	(-45.40, 0.001)	(-36.30, 0.001)	(15.40, 0.001)	(-30.10, 0.001)
$GMM-Bénézit$	(-16.50, 0.001)	(18.80, 0.001)	(-70.50, 0.001)	(-65.60, 0.001)	(17.70, 0.001)	(-147.6, 0.001)

3.6 Conclusion and future directions

We proposed an asynchronous distributed clustering algorithm framework for mesh networks, which respects data privacy, while balancing communication cost and clustering quality. Methods that use full k-means clustering at each node each cycle, and which exchange cluster shape and density descriptions, require fewer messages and give higher accuracy relative to centralised k-means, compared to previous methods. Distributed Gaussian Mixture Model clustering achieves high accuracy, and requires less elapsed time. The methods do, however, leak some privacy about the data sensed by each individual node. The results show that more informative cluster descriptions improve distributed clustering.

Note that we assume that the number of clusters to be described is known by all agents at the start, which is a strong assumption for the agents. In some practical

applications, agent may have different estimation of the number of clusters. If the number of clusters are fixed in advance, agents are forced to cluster their local data into predefined clusters. In next chapter, we will consider the case where the nodes are all sensing from the same pattern, but where the k-value is unknown. In addition, we only compare the accuracy against centralised method. It is possible that the centralised method does not fit well with the underlying datasets. We will consider the accuracy against ground truth as well in next chapter.

Chapter 4

Adaptive Asynchronous Distributed Clustering for Mesh Networks

In last chapter, we proposed an asynchronous distributed clustering algorithm framework for mesh networks. Agents do local clustering and share the summary descriptions. Eventually, all agents end up with the same understanding of the phenomenon. All algorithms that we developed assumed that the number of clusters is known by all agents in advance. This assumption is also adopted by most existing state-of-the-art work. However, in most practical applications, there is no prior information on how many clusters there should be. Therefore, an adaptive asynchronous clustering algorithm is required, which can derive the best number of clusters as part of the algorithm. We could simply rerun the algorithm from the previous chapter for each possible $\boldsymbol{k}$ value, but this would be expensive in both elapsed time and in communication cost. We note that one recent paper on distributed clustering, [Bendechache et al., 2016], does derive the appropriate k-value automatically, by starting with large $\boldsymbol{k}$ values, and then combining multiple clusters together as information flows up a tree in the network. We focus here on algorithms for a flat network. For comparison, we also consider methods which send all raw data to a single identified node.

Note that the network-wide model learned by all agents might not be identical (but are similar) due to the stochastic process of regeneration and clustering algorithm. However, in some practical applications, picking one specific model from all learned model is required. Motivated by this, we consider two new

assumptions about the network in this chapter: either that all agents may finish with slightly different models or all agents must finish with identical models.

The outline of this chapter is as follows: first, we present the research problem. Then in section 4.1 we discuss the adaptive distributed algorithm and requirements on the process: whether agents should all finish with the same model, or whether they can have different (but similar) models. In section 4.2, we describe the centralised methods, followed by introduction of measurement that will be used in section 4.5, where experimental results are given. Conclusion is summarised in section 4.7.

4.1 Distributed Algorithms

Since we relax the assumption that the number of clusters is known to agents in advance, now the question is that how to generate a consistent global cluster model if agents are not given the number of clusters $\boldsymbol{k}$ as input? In a distributed mesh network environment, the data collected by individual agents is only a fraction of the available data, and so it is not robust to infer the $\boldsymbol{k}$ value from just one agent's data. The approach we will develop starts with each agent inferring its own $\boldsymbol{k}$ value, and then adjusting this value as the local cluster models are shared.

As before, agents will use the regeneration method when merging cluster models. Now, in addition, the agents will use the silhouette method to infer $\boldsymbol{k}$ from that re-generated data. Algorithm 17 shows the process. First, an agent estimates the number of clusters locally. Then a summary description is shared with neighbours. Whenever new information is received, agent processes it in the following steps:

1. if it is a terminate message, chooses the centroids, counts and $\boldsymbol{k}$ value, that contained in the terminate message, as its final estimation. It then forwards this terminate message to its neighbours.

2. if it is a final summary description, it combines the regenerated data from all summary descriptions and estimates the $\boldsymbol{k}$ value again. Then, it clusters the combined data, computes the final summary, and sends out a terminate message to all its neighbours.

3. if it is a new summary description, there are two different options based on agents' knowledge about the network. When an agent is only aware

of its neighbours, it regenerates data from the summary description and combines it with its local data. Then it estimates the number of cluster $\boldsymbol{k}$ again and produces the new summary description by re-clustering. The new estimated $\boldsymbol{k}$ and centroids will be the assumed global view for agents while the agent waits for a termination message or some new information. If the size of network is known to agent, then the agent forwards this description to all neighbours.

Algorithm 17: Adaptive asynchronous distributed clustering algorithm

```
1  Agents estimate the k value locally ;
2  Agents run local clustering to completion ;
3  Agents transmit summaries with agent ID to neighbours;
4  while not stabilised or messages received do
5      if Receive terminate message then
6          Update local knowledge and k value;
7          Transmit terminate message to all neighbours;
8          Terminated;
9      else if Receive final summary then
10         Estimate the k value ;
11         Compute global view;
12         Attach a timestamp to this produced model;
13         Send converge command to neighbours;
14     else
15         if The size of network is not available then
16             Incorporate received neighbours' cluster information;
17             Run local clustering to completion;
18             Transmit new summaries to neighbours;
19         else
20             Restore the summary received and transmit it;
```

In addition to the two main scenarios we considered in the previous chapter, here we consider two new assumptions:

1. agents may finish with different models.
2. agents must finish with identical models.

The basic process for these two assumptions is as follows:

1. If we accept that agents could end up with slightly different models, then agent picks the model that is received or produced first as their final model and ignores any model that comes later.

2. If converging with identical models is required, all agents choose the model with the earliest timestamp as their final model. In this case, an agent will update its model only when the received model has an earlier timestamp than its previous model.

4.2 Centralised methods

On top of the two assumptions and two scenarios, we also consider two centralising methods. These two centralising methods are based on a tree based topology, which differ from the flat topology used in last chapter. Instead of sharing a basic model among neighbours, we consider two methods:

1. Centralise raw data points to a designated central agent. For individual sensor, it requires multiple packets to send out all its raw data points. We consider two different assumptions: in the first, there is no inter-packet delay, while in the second there is a uniform inter-packet delay of 0.1s.
2. Each agent produces its own basic model as before, and then transmits that model to the designated central agent.

To do this, we have to define the routing table for agents. We assume that this routing table is available for each sensor in advance and the computation cost to compute the routing path is ignored here. A shortest-path tree is constructed with the central agent as the root. For different type of nodes, the sharing strategy is listed as following:

1. Root node: wait until all information is received from its children, and then compute the final model.
2. Leaf node: send all messages to its parent node (raw data points or basic models).
3. Other node: send all its local data to its parent node and whenever messages are received from child nodes, relay the message on to the parent.

The applied protocol and the transmission of raw data has an impact on the communication measures. Information is transmitted in packets. For IEEE 802.15.4, the maximum size of a single packet is 128 bytes and the size could be increased to 11,452 bytes when IEEE 802.11ac is applied. In our previous chapter, we assumed that all abstract information could be sent by one packet, and the size is packet is around 10,000 bytes. The total size for 200 data points is 22,500 bytes,

so there are in total 3 packets to send.

Figure 4.1 shows different scenarios considered in this chapter. Note that when agents are only aware of their immediate neighbours but identical final model is required, the cost is expected to be expensive. If the size of network is not available, an agent has to compute its provisonal final model whenever a new information arrives, as it may be the last information it will receive. After each update, the model may has to be transmitted. For these reasons, we do not run experiments on this.

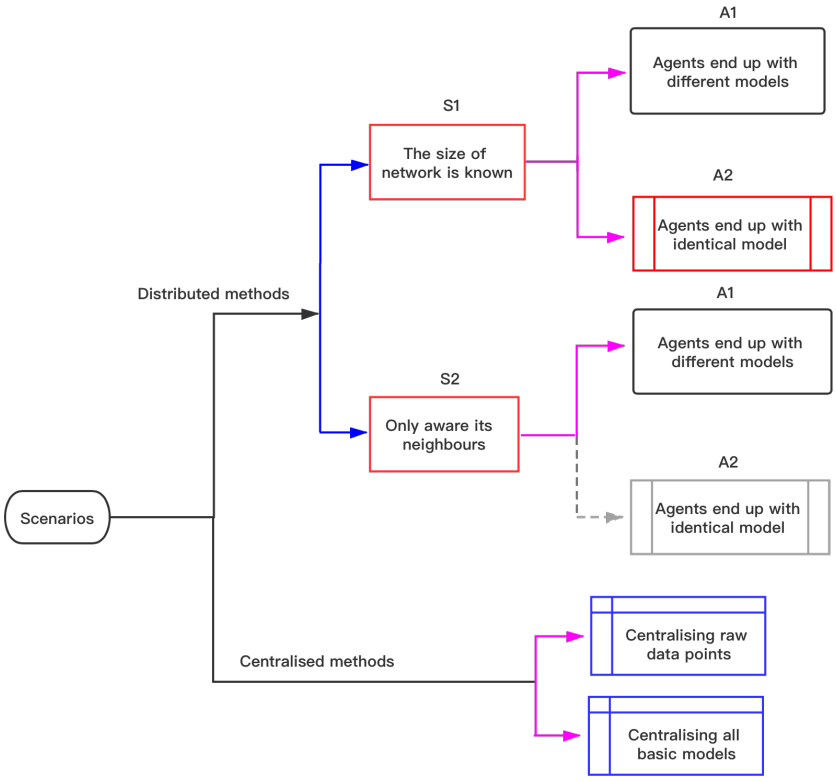


Figure 4.1: Different scenarios considered in this chapter

4.3 Time Complexity

The proposed distributed algorithm 17 consists of obtaining the initial cluster model, generating data points and computing the global models. Compared to the algorithms in last chapter, an agent has to decide the number of clusters $\boldsymbol{k}$ before obtaining the initial cluster model. When the number of cluster $\boldsymbol{k}$ is not pre-defined, the time complexity of silhouette method is bounded by $\mathcal{O}(N^2)$, where N is the number of data points.

When the size of network is known, the time complexity for the distributed algorithms, no matter which clustering algorithm is used, is bounded by $\mathcal{O}(N^2+N) = \mathcal{O}(N^2)$. If agent is only aware of its neighbours, the time complexity is bounded by $\mathcal{O}(N^2 + 2TN) = \mathcal{O}(N^2)$, where T is the number of rounds required to obtain the final cluster model.

The centralised method C_R consists of selecting the number of clusters $\boldsymbol{k}$ and doing a complete clustering. The overall time complexity for three clustering variants is bounded by $\mathcal{O}(N^2 + N) = \mathcal{O}(N^2)$. The centralise method C_B takes three steps: selecting the number of clusters $\boldsymbol{k}$, generating data points based on received local models, and computing the final model. It takes time in $\mathcal{O}(N^2 + N + N) = \mathcal{O}(N^2)$.

4.4 Measurement

In addition to the convergence time and communication costs, we also measure the number of learned clusters for all agents on average. The accuracy measurement used in the previous chapter measures how accurate each agent's model is on its own data compared to the centralised baseline or to the ground truth (original generation of the data points), and it is denoted by M1. However, this penalises the agent when its local data does not fit well with global model learned. Measuring the accuracy against all raw data is expensive and slow, so we use a new test dataset that sampled from the same ground truth distribution instead (denoted by M2).

Again, we consider a wireless network with 10 agents. Each node is given 200 data points and two types of underlying network topology are used: dense and sparse graph. Each message transmission delay is generated uniformly from the range [0.5, 1.0], and nodes begin their work at time randomly selected in [0, 0.1s]. The experiments are repeated for 20 runs to get a more accurate performance measurement. Three different underlying clustering algorithms are used: K-means, GMM and SG. Three different types of two-dimensional datasets are tested: fully symmetric multi-dimensional Gaussians, partially asymmetric multi-dimensional Gaussians, and rectangle uniformly distributed data. For each type of dataset, there are five generated clusters and three different types of clusters:

1. Heavily-overlapped clusters
2. Slightly-overlapped clusters

3. Well-separated clusters

Figure 4.5 shows an example of different overlaps between cluster, and the notation is summarised in Table 4.1.

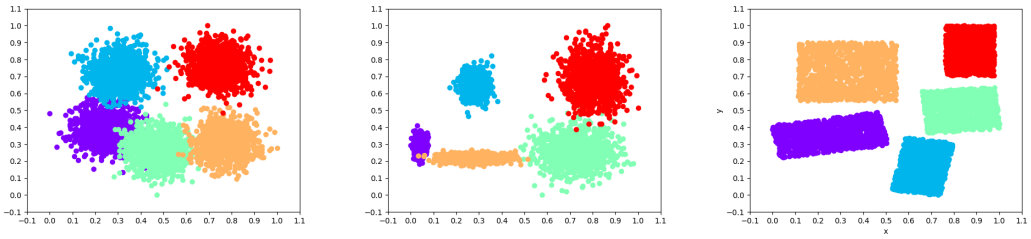


Figure 4.5: An example of different overlaps between clusters

Table 4.1: Notation of different methods and measurements

S1_A1(1)	The size of network is known and agents may finish with different models, the first agent who received all message shared its produced model with neighbours
S1_A1(2)	The size of network is known and agents must finish with different models, the first agent who received all message shared its latest information with neighbours
$S1_A2$	The size of network is known and agents must finish with identical model
$S2$	Agents only aware their neighbours and may finish with different models
C_R	Centralise all raw data point to a central agent
C_B	Centralise basic models to a central agent
$Accuracy_c$	Accuracy against centralised clustering method
$Accuracy_g$	Accuracy against ground truth
$M1$	Evaluate the performance on agent's own data only
$M2$	Evaluate the performance on a new test data

4.5 Experimental Results

We only show the experimental results for dense graphs here; the results for result on sparse networks are attached to the appendix 8.2. The difference between dense networks and sparse networks lies in the amount of transmitted message. Few transmissions are required when the networks are sparse, because each agent has fewer neighbours to share information with. We will discuss the main results with each figure, and a table of significant tests will be given at the end.

4.5.1 K-means as clustering algorithm

First, we measure the performance when K-means is used as the local clustering algorithm. Figure 4.6 shows the comparison result when the underlying network is dense. There are significant differences in convergence time (except $S1_A1$ and C_B) and message transmission between centralised methods and distributed methods.

Centralising raw data points converges faster than other methods when inter-package delay is not considered. Centralised basic model C_B requires fewest message transmissions. Since we assume that the routing table is known in advance and a shortest-path based tree is used, we expect this method will show the least network overhead. However, the standard deviation shows that some agents relayed many messages while other agents did not.

For distributed methods, $S1_A1$ converges faster than other methods. This is as expected because $S1_A1$ assumes that the size of network is known and agents could end up with different models, where agents do not need to keep updating their models and sending more packets, and that fewer communications are needed to share the final models. $S2$ requires the largest amount of message transmissions, followed by $S1_A2$. The most likely reason behind this is each agent is only aware of its immediate neighbours under $S2$ and has no idea about the size of network. Agents in S2 are repeatedly updating their models and transmitting them again, while in $S1_A2$, if multiple final models are created independently, agents may be forced to accept updates with lower timestamps and so have to retransmit a final model to their neighbours.

The extent of overlapping of clusters affects how accurate the number of learned clusters is, compared to the ground truth. The third row of figure 4.6 shows that centralised methods predict a more accurate number of clusters than the distributed methods for the overlapped cases. For the well separated case, they are all getting approximately 5. The last four rows show the accuracy against centralised method and ground truth, measured by M1 and M2 separately. Centralising all raw data points C_R achieves the highest accuracy against ground truth. That is as expected. C_R receives all the raw data unfiltered, while the other methods are summarising, regenerating and re-clustering, and so can be expected to introduce errors.

When accuracy against the centralised method is measured with an agent's own data M1, centralising all basic models C_B achieves the highest accuracy. However distributed method $S1_A1$ performs better than C_B when the same accuracy is measured with a new test data M2. It indicates that agent's local data may not fit the learned model very well in some cases. Note that when accuracy against ground truth is measured, there is very little difference between M1 and M2.

Figure 4.7 shows the performance when data points that generated from partially asymmetric Gaussians are used. Compared to the centralised methods, there is a

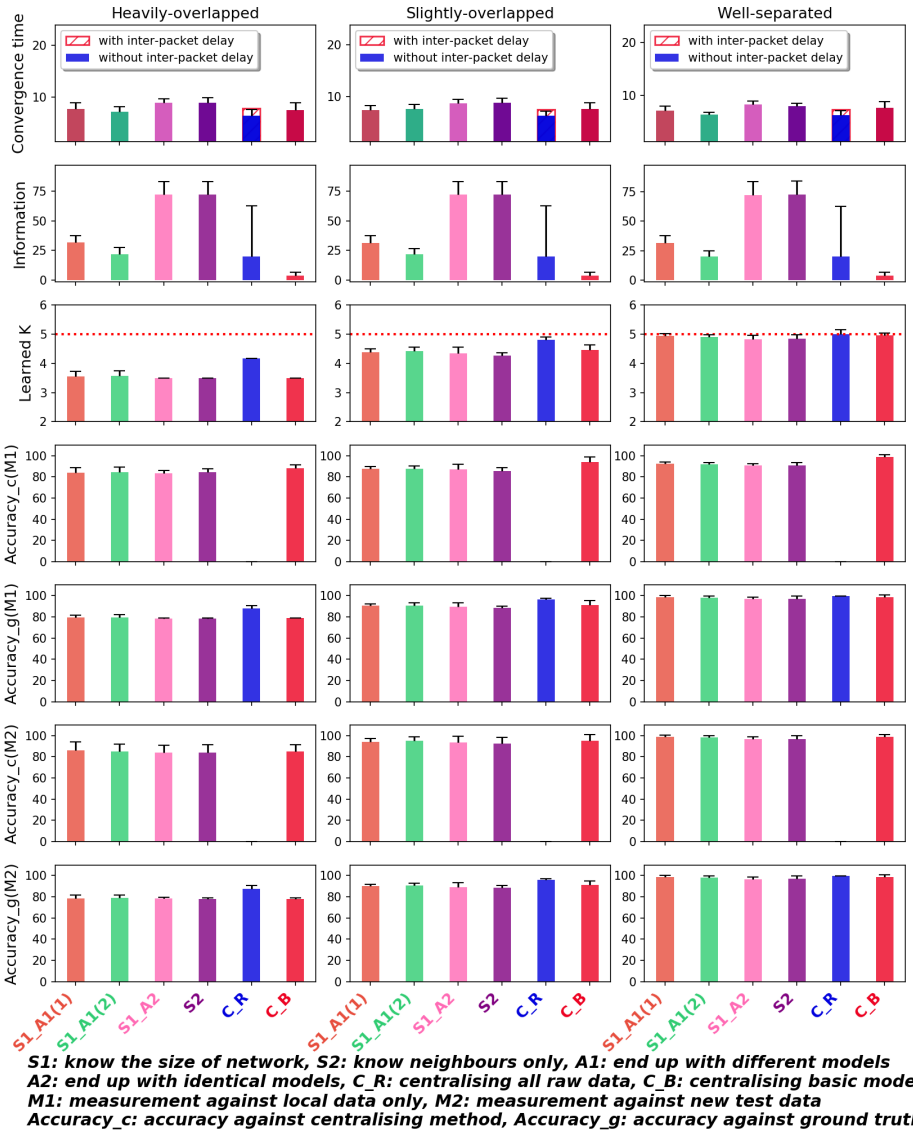


Figure 4.6: K-means with symmetric multi-dimensional Gaussians on dense network

significant difference in message transmission, but there is no significant difference in convergence time between C_B and distributed methods.

Compared to that with symmetric Gaussians, the accuracy against data points and accuracy of the inferred number of clusters has dropped. However, the convergence time and total message transmission remains unchanged. Note that when the clusters are well separated from each other, there is no difference in accuracy against data points and accuracy of the inferred number of clusters. That is because K-means is able to cluster data points when clusters are well separated.

Figure 4.8 shows the performance when uniformly distributed datasets are tested. The message transmission by distributed methods differs substantially from centralised methods, and there is a significant difference in convergence time between C_R and distributed methods.

Again, the convergence time and the number of transmitted messages remains stable. Compared to the previous two figures, accuracy of the inferred $\boldsymbol{k}$ is poorer. In addition, accuracy on data points against ground truth (both M1 and M2) has dropped significantly. This is as expected because it is a challenge for K-means to handle uniformly rectangle distributed data. If the data is uniform inside a long rectangle, that K-means will struggle, because points at the end of one rectangle may be closer to the centroids of another. Compared to accuracy on data points against centralised method (the fourth and sixth row), the accuracy on data points against ground truth (the fifth and seventh row) is much lower. It indicates that simply comparing to centralised method is not reliable when clustering algorithm is not fitting well with the underlying dataset.

To summarise, centralised method C_R takes the shortest time to converge if inter-packet delay is not considered, and the convergence time depends on what value you use in the simulator for inter-packet delay. Centralised method C_B requires fewest transmissions. However, the high standard deviation shows that agents close to the root have to relay many packets while agents far away from the root only forward a few packets.

In terms of the number of learned clusters, the extent of overlapping between clusters plays a vital role. When clusters are well separated from each other, our proposed algorithm matches the ground truth number of clusters perfectly, no matter what the underlying dataset and network topology are.

When K-means is used as clustering algorithm, it achieves the highest accuracy on

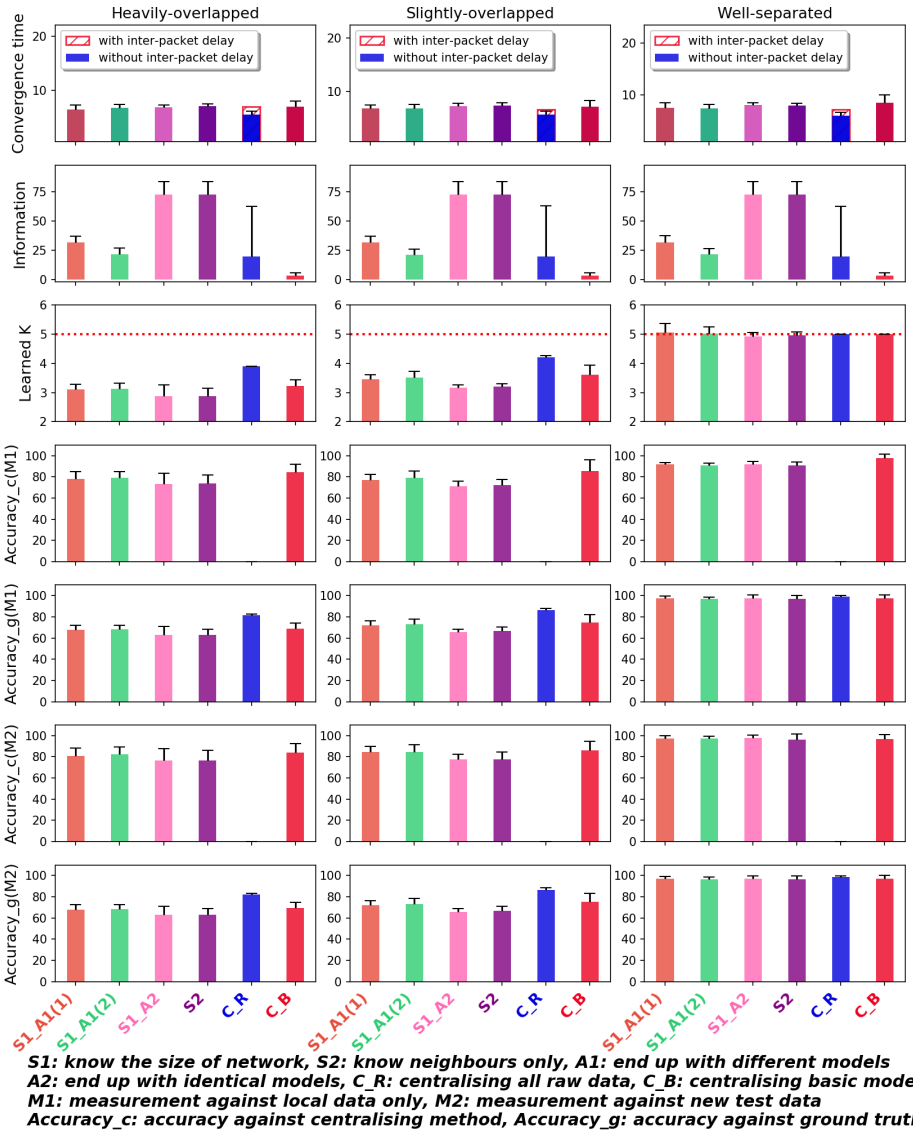


Figure 4.7: K-means on dense network with asymmetric multi-dimensional Gaussians

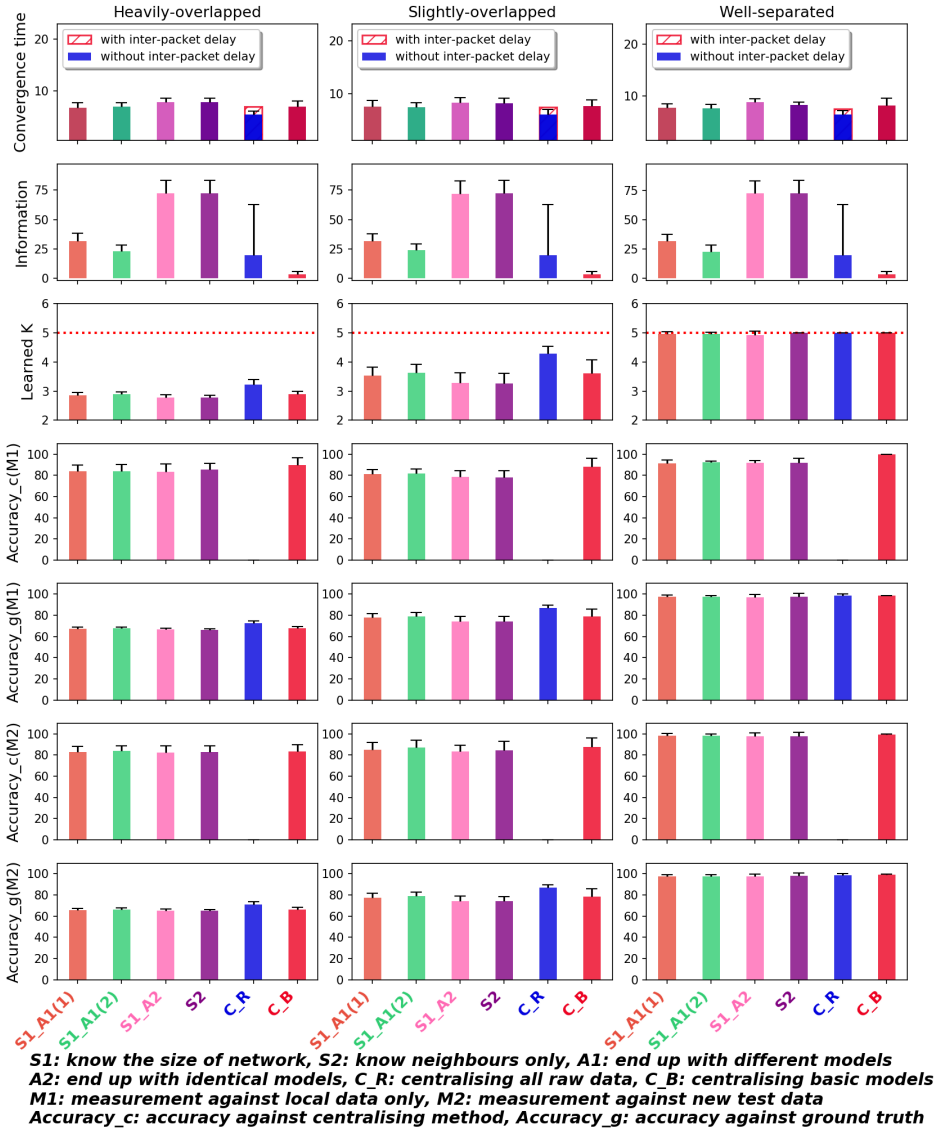


Figure 4.8: K-means on dense network with uniform distributed datasets

data points against centralised algorithm and ground truth when the underlying datasets are formed from symmetric Gaussians clusters. The accuracy on data points against ground truth drops to about 70% when partially-symmetric Gaussians are used, then declines to around 65% when uniformly distributed clusters are used. However, the accuracy on data points against the centralised algorithm remains stable even when the underlying dataset appears a challenge for K-means to cluster. It indicates that the drop in accuracy is due to the underlying K-means algorithm, and not the distributed protocol.

4.5.2 SG as clustering algorithm

We now measure the performance when K-means is used as clustering algorithm, but where each cluster is then fitted by a separate Gaussian for using in information sharing. In this case, the shape of clusters is described by the mean and variance, rather than just the mean (i.e. the centroids) only.

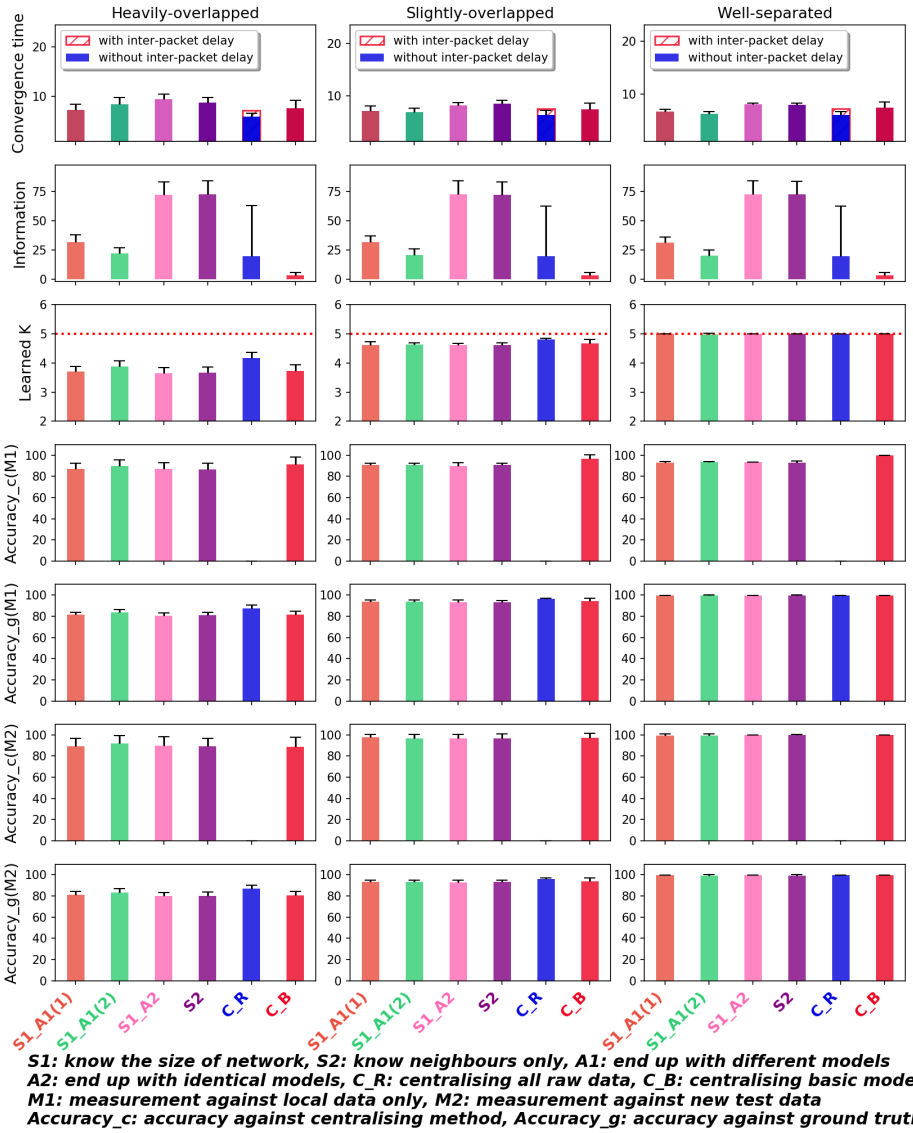


Figure 4.9: SG on dense network with symmetric multi-dimensional Gaussians

Figure 4.9 shows the result on dense networks when fully symmetric multi-dimensional Gaussians are used to generate the raw data. As before, there are significant differences in message transmission and convergence time (except $S1_A1$

against C_B) between centralised methods and distributed methods.

Again, centralising all raw data points to a central agent C_R converges faster than other methods if inter-packet delay is not considered. Moreover, the number of learned clusters, accuracy on data points against centralised algorithm and ground truth appear to be better when the extent of overlapping between clusters reduces from heavily overlapped to well separated, reaching accuracy levels over 80%.

Figure 4.10 shows the performance of different methods on dense networks when partially-symmetric multi-dimensional Gaussians is tested. Similarly, there are significant differences in message transmission and convergence time (except distributed methods against C_B) between centralised methods and distributed methods. Compared to that with symmetric Gaussians, accuracy on data points against ground truth in both M1 and M2 dropped.

Figure 4.11 shows the comparison result of different methods on dense networks with uniformly distributed ground truth cluster data. Compared to the centralised methods, there is a significant difference in message transmission. Since it is already a challenge for K-means to handle uniformly rectangular clusters, as expected, the accuracy on data points against ground truth is lower than that with symmetric or asymmetric Gaussians.

Similar to K-means, when SG is used to summarise the clusters that were found by K-means: 1), centralised method C_R converges faster than other methods if inter-packet delay is not considered 2), centralised method C_B requires fewest transmissions. 3), there is little difference in the number of learned clusters, accuracy on data point against centralised method, accuracy on data points against ground truth when clusters are well separated. 4), SG preforms better when underlying datasets are formed from fully symmetric Gaussians clusters and there are overlaps between clusters.

4.5.3 GMM as clustering algorithm

In this section, we measure the performance when GMM is used as clustering algorithm. In this case, the collection of clusters is described as a collection of Gaussians. For three different underlying datasets, there are significant differences in message transmission and convergence time between centralised methods and distributed methods.

Figure 4.12 shows the result on dense networks with data points formed from

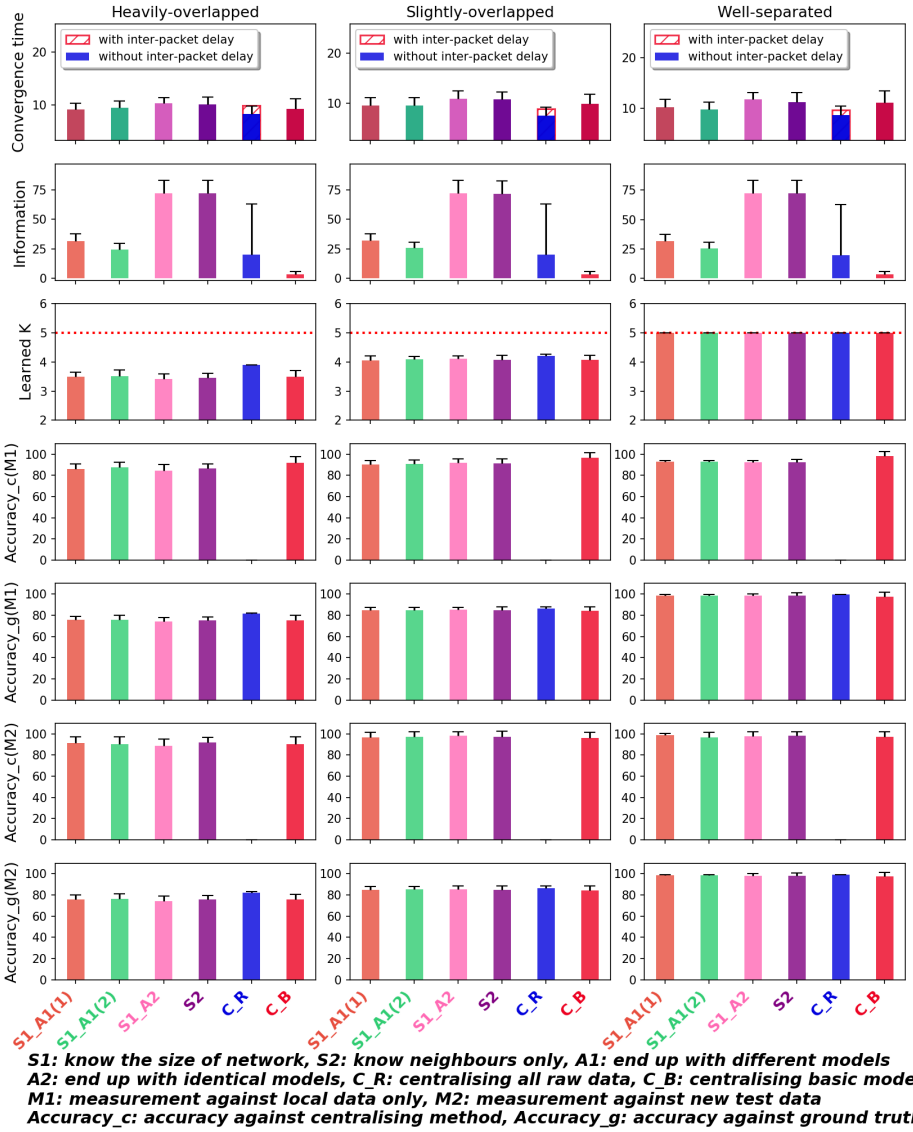


Figure 4.10: SG on dense network with asymmetric multi-dimensional Gaussians

fully symmetric Gaussians. Again, centralising all data points C_R takes the shortest time to converge when inter-packet delay is ignored. In addition, C_B requires the least amount of message to transmit. All centralised methods and distributed methods achieved really high accuracy, even when there are heavy overlaps between clusters.

Figure 4.13 shows the result on dense graphs when partially symmetric Gaussians are tested. Compared to Figure 4.12, there is little difference between the accuracy on data points against ground truth and centralised algorithm when

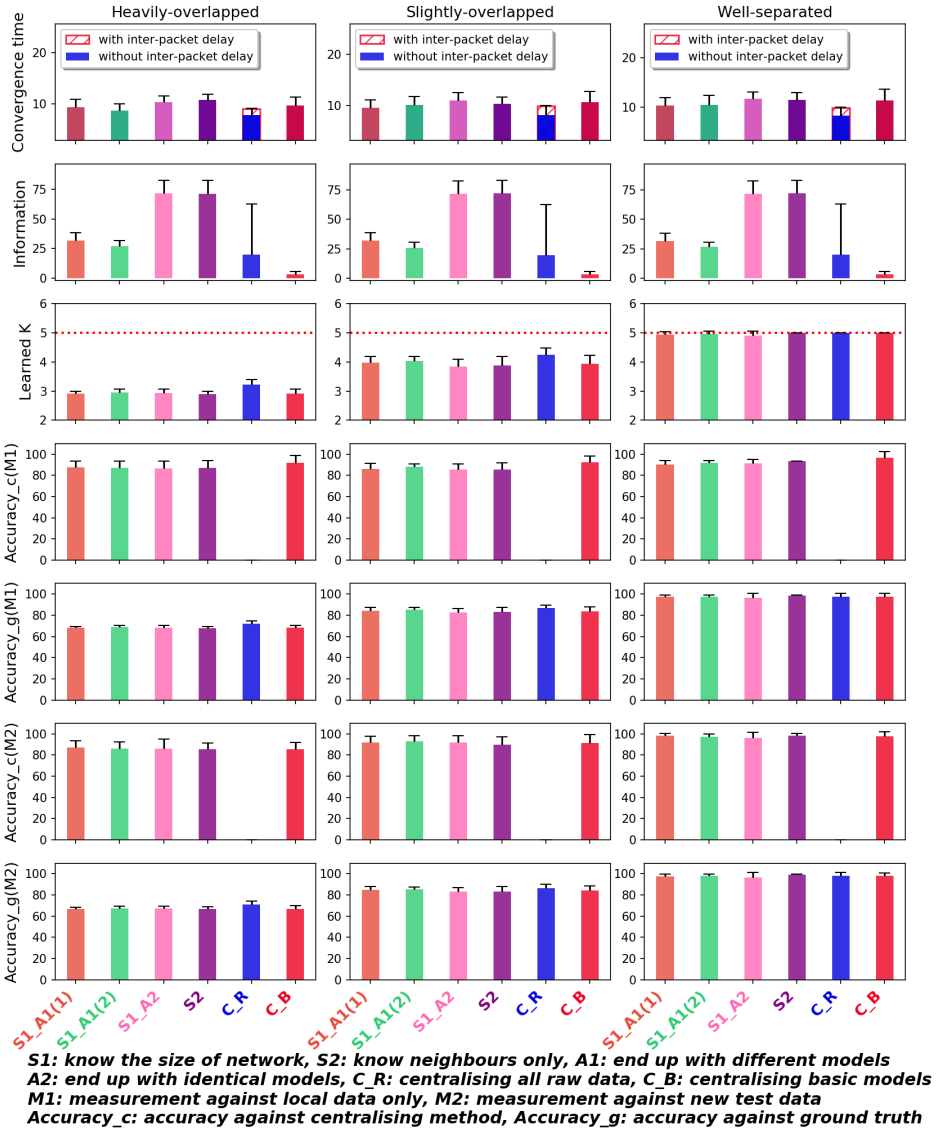


Figure 4.11: SG on dense network with uniform distributed dataset

clusters are well separated from each other. However, when the extent of overlapping increased, the accuracy against ground truth dropped sharply while the corresponding accuracy against centralised algorithm dropped by less than 10%.

When uniformly distributed data is used as underlying dataset, we expected that accuracy against ground truth would be lower than that with symmetric or asymmetric Gaussians. Figure 4.14 shows that the accuracy against ground truth is around 62% for both M1 and M2. However, the accuracies against centralised algorithm for M1 and M2 are still high, more than 80%.

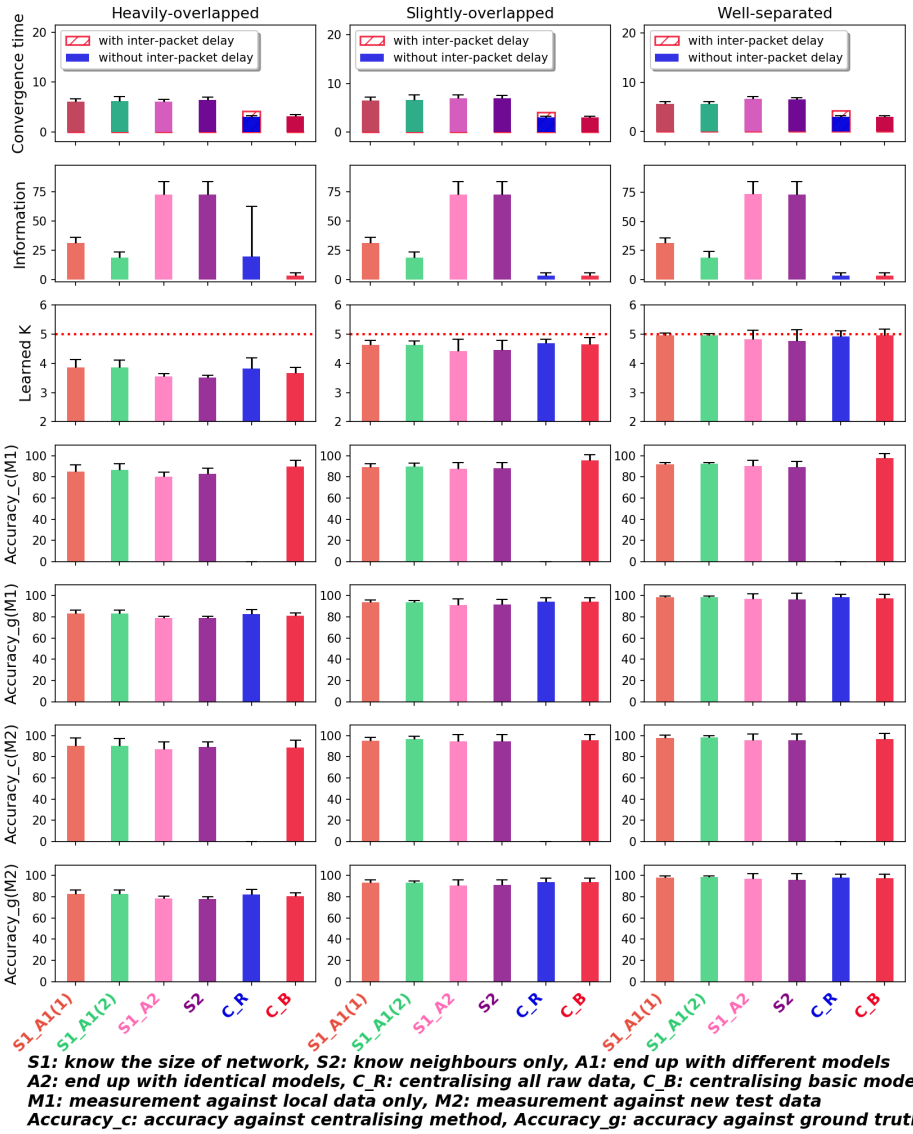


Figure 4.12: GMM on dense network with symmetric multi-dimensional Gaussians

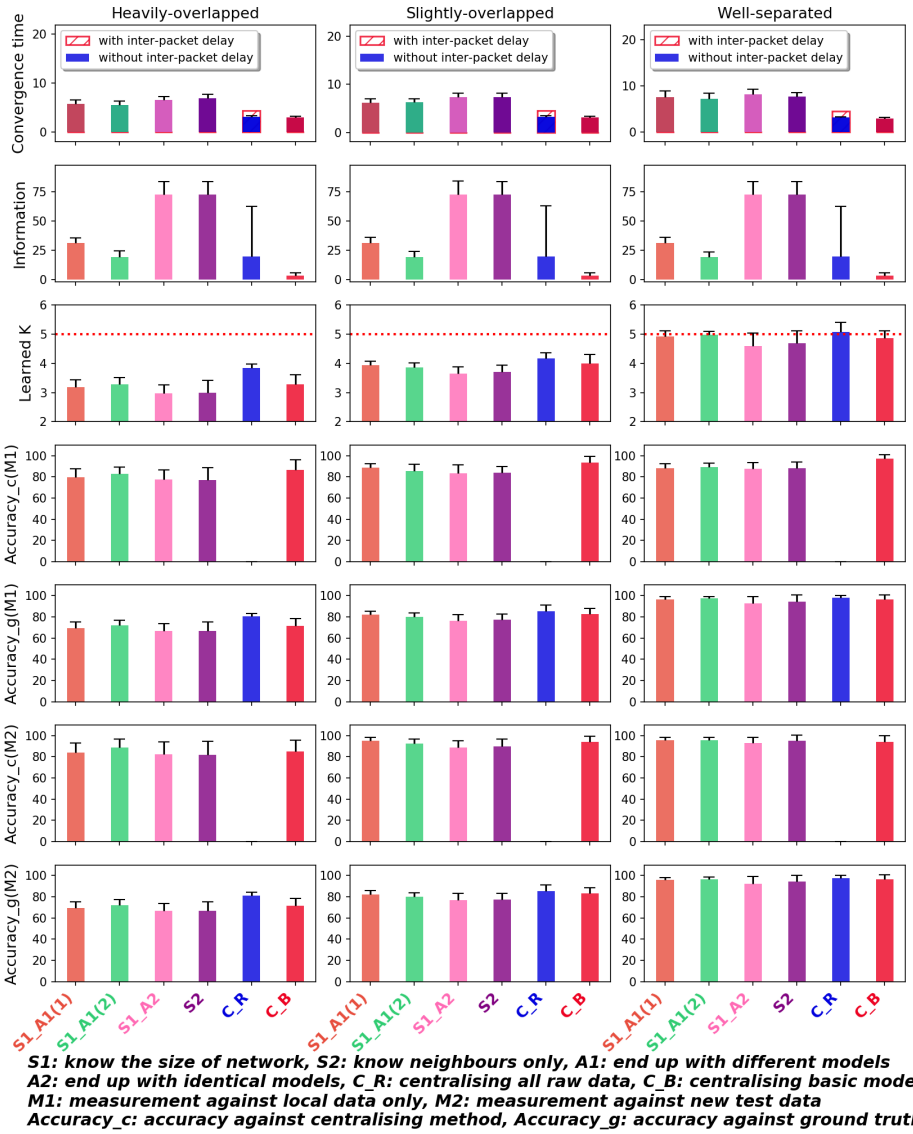


Figure 4.13: GMM on dense network with asymmetric multi-dimensional Gaussians

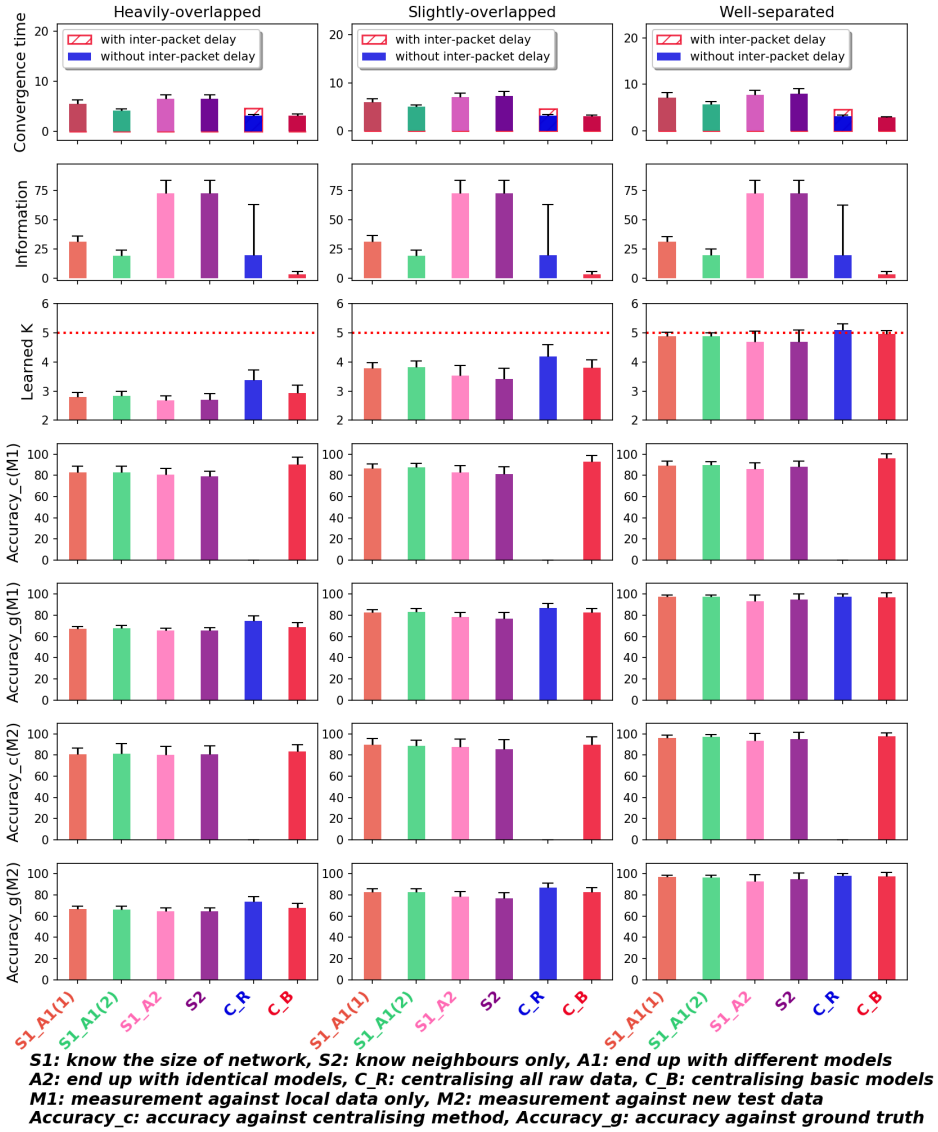


Figure 4.14: GMM on dense network with uniform distributed dataset

Compared to K-means and SG, the result is similar when GMM is used as local clustering method. The centralised method C_R converges slower than other methods if inter-packet delay is considered and centralised method C_B requires fewest transmissions. When clusters are well separated from each other, our proposed algorithm learned the perfect number of clusters, achieved nearly perfect accuracy against centralised algorithm and ground truth, no matter what the underlying dataset and network topology are.

4.5.4 Summary

In this section, we outline the difference between these three clustering algorithm with different underlying datasets and network topologies. In terms of convergence time, GMM convergences faster than SG and K-means. That is because K-means has to compute the principal orientations of all found clusters. As expected, all methods perform nearly perfectly when clusters are well separated, no matter what the underlying datasets and network topologies are. So we only describe the difference in accuracy on data points when there are overlaps between clusters.

When underlying datasets are formed from symmetric Gaussians, we expected that all methods perform well. Figure 4.6, Figure 4.9 and Figure 4.12 show that accuracy on data points against ground truth achieved by GMM is the highest, followed by SG. The accuracy achieved by K-means is the lowest. When the underlying datasets are generated from symmetric Gaussians, mean and standard deviation would describe the cluster much faithfully than multiple bounding boxes. Figure 4.15 shows the difference between these two regenerations when the underlying dataset is generated by a fully symmetric Gaussian. The figure on the left shows the dataset generated from a symmetric Gaussian and its corresponding bounding boxes. Figure in the middle and right show the data points, that generated by bounding boxes and mean and standard deviation separately. Obviously, data points generated by mean and standard deviation are more similar to the original dataset.

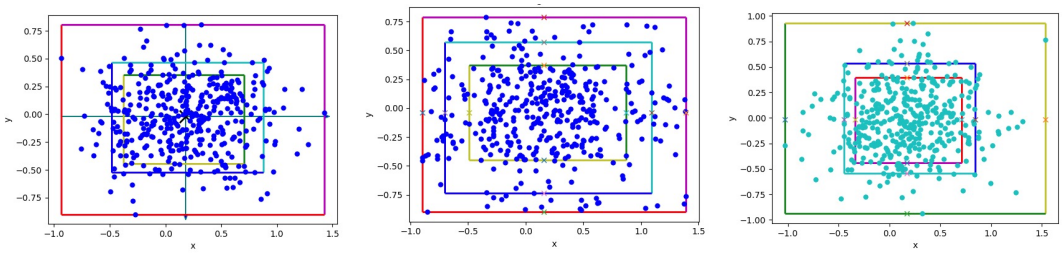


Figure 4.15: Comparison of different regenerations when underlying dataset is generated by a symmetric Gaussian

With asymmetric Gaussians, GMM should perform better than SG and K-means in accuracy against ground truth. When there are heavy overlaps between clusters (the first column of each figure), SG achieves higher accuracy against centralised algorithm and ground truth than K-means and GMM. When there are only slight overlaps between clusters (the second column of each figure), both SG and GMM achieved similar accuracy against centralised algorithm, but higher than K-means.

However, the accuracy against ground truth achieved by SG is higher than GMM and K-means.

For uniformly distributed dataset, none of these methods could partition the dataset into reasonable clusters. When there are heavy overlaps between clusters (the first column of each figure), there is little difference in terms of accuracy against centralised algorithm and ground truth. However, with slight overlaps between clusters (the second column of each figure), accuracy against centralised algorithm and ground truth by SG is the highest, followed by GMM. K-means performs the worst in these methods. Figure 4.16 shows an example of comparison of different regenerations when underlying dataset is generated by a rectangle uniform distribution. Apparently, data points generated by bounding boxes are similar to the ground truth distribution when original distribution is uniform.

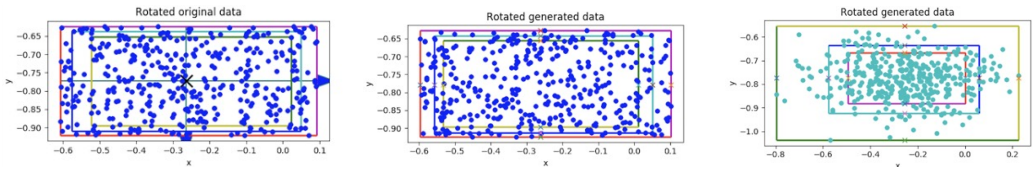


Figure 4.16: Comparison of different regenerations when underlying dataset is generated by a uniformly rectangle distribution

Centralised method C_R takes the shortest time to converge if inter-packet delay is ignored and achieves the highest accuracy against centralised algorithm and ground truth. Centralised method C_B requires the fewest transmissions. Compared to centralised methods, the amount of message required for distributed methods could reduce to a large extent if the size of network is known and agents may finish with different models. In terms of accuracy on data points against centralised algorithm, there is little difference between centralised methods and distributed methods.

4.6 Significance Study

In this section, we show the significance study of the experimental results obtained by centralised methods and distributed methods. A statistical comparison is thought to be significant when its P-value is less than 0.05. Note that the experiments are repeated for 20 runs, and it is necessary to test the assumption of normality. Before doing the significant test, we use the Shapiro-Wilk's test to assess whether the assumption of normality is hold.

Table 4.2 shows the result of normality test on convergence time and information

transmission. A testing sample is considered to be normal if the result by Wilk's test is not significant. The result shows that, for all scenarios, the difference is not significant with P-values all larger than 0.05. It indicates that the distribution of obtained result is normal. We omit the result of the normality test on Learned K and accuracy against ground truth, since the assumption of normality is not hold.

Table 4.2: Test for normality with Shapiro-Wilk's test, only the result with K-means as clustering algorithm is given

	Time			**Information**		
K-means	Symmetric P value	Asymmetric P value	Uniform P value	Symmetric P value	Asymmetric P value	Uniform P value
C_B	0.81	0.14	0.20	1.0	1.0	1.0
C_R	0.53	0.28	0.36	1.0	1.0	1.0
$S1_A1$	0.36	0.89	0.22	0.36	0.40	0.82
$S1_A2$	0.40	0.52	0.99	0.37	0.22	0.08
$S2$	0.58	0.93	0.63	0.68	0.56	0.79

Tables 4.3 shows the measurement on convergence time and message transmission. In term of convergence time, there is significant difference between centralised methods and distributed methods except some cases of $S1_A1$ and C_B. The possible reason behind this is that the basic procedure is similar: the first agent or central agent, who received all message from other agents, compute the global model and send to neighbours. When consider the message transmission, all P-value is less than 0.05, which shows there is significant difference between them.

4.7 Conclusion

If we assume that there is no inter-packet delay (t=0s), centralising all raw data requires the transmission of less information and the shortest convergence time. Centralising all raw data points to a central agent achieves higher accuracy on data points than centralising basic models and distributed methods when there is slight-overlapping or heavy-overlapping between clusters. However, this method suffers from the problem of single point of failure and leakage of data privacy. The standard deviation of communication cost is high, which means some agents have relayed huge amounts of information while others have not. This will drain the energy of agents and frequently recomputing the route table will become an issue.

In terms of communication cost, the figures above show that centralising basic

Table 4.3: Pairwise Tukey test for distributed methods and centralized methods, only the results on dense graph is compared(1). $\diamond$ denotes the difference of mean

	Symmetric Gaussians			**Asymmetric Gaussians**		
Time	K-means ($\diamond$, P value)	GMM ($\diamond$, P value)	SG ($\diamond$, P value)	K-means ($\diamond$, P value)	GMM ($\diamond$, P value)	SG ($\diamond$, P value)
$C_R - C_B$	(-1.30, 0.001)	(0.01, 0.9)	(-1.13, 0.001)	(-1.52, 0.001)	(0.15, 0.9)	(-2.35, 0.015)
$S1_A1 - C_R$	(1.07, 0.007)	(2.28, 0.001)	(0.823, 0.001)	(1.16, 0.016)	(2.91, 0.001)	(2.07, 0.04)
$S1_A2 - C_R$	(2.38, 0.001)	(3.98, 0.001)	(1.84, 0.001)	(1.66, 0.001)	(4.05, 0.001)	(3.38, 0.001)
$S2 - C_R$	(2.51, 0.001)	(3.94, 0.001)	(2.19, 0.001)	(1.73, 0.001)	(4.04, 0.001)	(3.21, 0.001)
$S1_A1 - C_B$	(-0.23, 0.9)	(2.30, 0.001)	(-0.31, 0.70)	(-0.37, 0.8)	(3.06, 0.001)	(-0.28, 0.9)
$S1_A2 - C_B$	(1.09, 0.006)	(3.99, 0.001)	(0.70, 0.04)	(0.13, 0.9)	(4.20, 0.001)	(1.03, 0.56)
$S2 - C_B$	(1.21, 0.002)	(3.95, 0.001)	(1.06, 0.001)	(0.20, 0.9)	(4.19, 0.001)	(0.86, 0.70)

	Uniform distributions		
Time	K-means ($\diamond$, P value)	GMM ($\diamond$, P value)	SG ($\diamond$, P value)
$C_R - C_B$	(-1.57, 0.001)	(0.18, 0.9)	(-2.45, 0.004)
$S1_A1 - C_R$	(1.46, 0.002)	(2.76, 0.001)	(1.36, 0.03)
$S1_A2 - C_R$	(2.24, 0.001)	(3.81, 0.001)	(2.82, 0.001)
$S2 - C_R$	(2.09, 0.001)	(4.02, 0.001)	(2.18, 0.01)
$S1_A1 - C_B$	(-0.10, 0.9)	(2.94, 0.001)	(-1.10, 0.43)
$S1_A2 - C_B$	(0.67, 0.365)	(3.99, 0.001)	(0.36, 0.9)
$S2 - C_B$	(0.53, 0.59)	(4.20, 0.001)	(-0.28, 0.9)

	Symmetric Gaussians			**Asymmetric Gaussians**		
Transmission	K-means ($\diamond$, P value)	GMM ($\diamond$, P value)	SG ($\diamond$, P value)	K-means ($\diamond$, P value)	GMM ($\diamond$, P value)	SG ($\diamond$, P value)
$C_R - C_B$	(39.25, 0.001)	(39.25, 0.001)	(39.25, 0.001)	(39.25, 0.001)	(39.25, 0.001)	(39.25, 0.001)
$S1_A1 - C_R$	(-11.28, 0.001)	(-11.43, 0.001)	(-11.17, 0.001)	(-11.08, 0.001)	(-11.44, 0.001)	(-11.01, 0.001)
$S1_A2 - C_R$	(29.37, 0.001)	(29.75, 0.001)	(30.04, 0.001)	(29.87, 0.001)	(29.87, 0.001)	(29.15, 0.001)
$S2 - C_R$	(29.30, 0.001)	(29.75, 0.001)	(29.39, 0.001)	(29.60, 0.001)	(29.81, 0.001)	(28.88, 0.001)
$S1_A1 - C_B$	(27.97, 0.001)	(27.80, 0.001)	(28.07, 0.001)	(28.17, 0.001)	(27.80, 0.001)	(28.24, 0.001)
$S1_A2 - C_B$	(68.62, 0.001)	(69.0, 0.001)	(69.29, 0.001)	(69.12, 0.001)	(69.13, 0.001)	(68.40, 0.001)
$S2 - C_B$	(68.55, 0.001)	(69.0, 0.001)	(68.64, 0.001)	(68.86, 0.001)	(69.07, 0.001)	(68.13, 0.001)

	Uniform distributions		
Transmission	K-means ($\diamond$, P value)	GMM ($\diamond$, P value)	SG ($\diamond$, P value)
$C_R - C_B$	(39.25, 0.001)	(39.25, 0.001)	(39.25, 0.001)
$S1_A1 - C_R$	(-11.15, 0.001)	(-11.44, 0.001)	(-11.03, 0.001)
$S1_A2 - C_R$	(29.12, 0.001)	(29.60, 0.001)	(28.63, 0.001)
$S2 - C_R$	(29.49, 0.001)	(29.57, 0.001)	(29.36, 0.001)
$S1_A1 - C_B$	(28.09, 0.001)	(27.80, 0.001)	(28.24, 0.001)
$S1_A2 - C_B$	(68.36, 0.001)	(68.85, 0.001)	(67.88, 0.001)
$S2 - C_B$	(68.75, 0.001)	(68.82, 0.001)	(68.61, 0.001)

models to a central agent performs much better than other methods. This method preserves the data privacy but still suffers from the problem of single point of failure. Note that we assume that the shortest path is available for each agent in advance and we ignore the cost to compute the routing table. Since agents in wireless mesh networks are usually powered by limited batteries, this assumption is not appropriate.

The extent of overlapping between clusters has a big effect on the learned number of clusters. When clusters are well separated from each other, all proposed meth-

ods learn the same number of clusters as ground truth. For distributed methods, more information is required to be transmitted if it is required that agents must finish with identical models. However, if we accept that agents may finish with different models, it requires much less information transmission and much less elapsed time. Compared to the highest accuracy of the centralised method, the clustering accuracy achieved by the distributed method dropped by around 9% and the communication cost increased by up to 20%. But the convergence time has been reduced and the problem of a single point of failure has been avoided. In addition, we respect the data privacy. The distributed method offers similar clustering accuracy to the centralised methods but relaxes the assumption that agent knows the whole routing path in advance.

Note that we assume that each agent is sensing data for the same phenomenon in this and the last chapter. In many practical applications, group of agents may sense data from different sources. For instance, agents near the fire may have different temperature readings from others. We should separate these two readings into different models, rather than simply combining these two readings into one model. The question is then how to ensure the agents in the network can recognise that they are sensing different phenomena, and can assign themselves (i.e. their data) into the correct subgroups. We will address this problem in the coming chapters.

Chapter 5

Identifying multiple patterns in distributed clustering

In previous chapters, we focused on clustering algorithm in wireless mesh networks, where each agent is sensing data from the same environment. Each agent represents its raw data points as a cluster model (the number of clusters are not necessarily consistent), and shares this model with its neighbours. Agents must combine the shared models into a single description, and eventually, all agents end up with the same understanding of the phenomenon (similar but not identical).

In this chapter, we extend the problem to consider the case where sub-groups of agents receive data from different environments, and so the inferred cluster models should be different for each sub-group (and could have different numbers of clusters in each). We define these different environments as patterns. First, to measure the similarity or difference between two cluster models, we propose a method based on the Earth Mover's Distance (EMD). We can then use this to search for clusters of similar agents, which we infer are seeing the same pattern. Then we go back to the general question: how should the agents act and communicate so that all agents understand the environment (for instance, how many patterns observed, and do agents know what other agents are in their patterns, etc) when there are group of agents receiving data from different environments.

The outline of this chapter is as follows. In the first section 5.1, we describe the research questions and the main algorithm used to address the questions. We introduce a EMD based method to measure the similarity between agents and show the performance in section 5.2. After that, new requirements and sharing strategies are described in section 5.3. The measurement for multiple pattern

networks are illustrated in section 5.4 and experimental results are shown in section 5.6 and 5.7. Conclusion is given in section 5.9.

5.1 Research question and proposed solutions

Suppose in a n-agent mesh network, m agents ($m \leq n$) are sensing data from one phenomenon and some other agents are receiving data from different phenomena. We call these phenomena "patterns" to be detected in the network, and we note that they are different from the sampling variation that may occur for individual agents observing the same phenomenon. Is it possible for the distributed clustering algorithm to predict the right number of patterns and identify those patterns, while preserving high clustering accuracy within each pattern? Predicting the wrong pattern for agents, for instance, assigning an agent to a sub-pattern that differs from the ground truth sub-pattern that it came from, may affect the clustering accuracy and estimation of the number of clusters.

Compared to the procedures in preceding chapters, an agent has to run a similarity test on the models whenever it is expected to combine multiple models into one or more, or to update a model with some new information. Based on the similarity measurement, it infers the number of patterns. If there are more than one pattern, this agent runs separate processes (for instance, clustering, updating and regenerating) for each pattern. Throughout this chapter, we will use similarity between agents to mean similarity between their cluster models.

First, we focus on the similarity measurement. As we mentioned before, this step plays an important role in inferring the number of patterns and group assignment of all agents. The question is how to measure the similarity of cluster models learned by different agents? Note that the number of clusters inside the model learned by different agents may be different, so the widely used L1-norm and L2-norm Euclidean distance can no longer be used. To address this problem, we propose an EMD-based distance metric to measure the similarity between agents. Although there are several advantages of EMD (robust to noise and globally shape-aware), it is not widely used outside the Computer Vision area due to the high computation cost. To mitigate the high computation cost, we propose that an adapted EMD metric that is applied to the cluster models (centroids and corresponding count), rather than to the raw data points.

Then we go back to the general question, how to understand the environment when there are multiple patterns. Compared to algorithms in the preceding

chapters, there are several differences. If an agent receives a terminate message, it will accept the patterns and group assignment received as its estimation of patterns and group assignment. However, when the size of the network is not known (S2), an agent has to compare the similarity between clusters models received and its own whenever it receives a new model. The agent will integrate its models only when it believes the agent who sent the message is similar. When agents know the size of network, the agent who received all cluster models will compare the similarity between agents and infer the patterns.

In previous chapters, we were concerned with accuracy of data clustering. We assumed that the raw data that the algorithm operates on is generated by some underlying phenomenon or data distribution. After the algorithm completes, we evaluate accuracy in terms of data point co-clustering - given new data points sampled from the distribution, how well do our agents cluster the points in accordance with the underlying distribution (or by comparison to a single central clustering algorithm). The previous chapters assumed a single pattern for all agents - that is, all agents observed data sampled from the same distribution. In this chapter, we assume multiple patterns - that is, in ground truth, one sub-group of agents observes data from one distribution, while other sub-groups observe data from other distributions. A ground truth pattern is thus a data distribution plus a group of agents that sample from it. The aim of the algorithm is now to assign agents to groups, and to infer the data distribution for each group. For an individual agent, we can assess data accuracy as before - does its model cluster data in accordance with the ground truth distribution that generated the data. But we now need an additional measure - are the agents assigned into groups or patterns in the same way as the original ground truth (or, as before, in the same way as a single centralised algorithm). This definition and evaluation of this new accuracy against patterns will be discussed further in Subsection 5.4, and will be defined precisely for each different scenario.

5.2 An EMD based method to measure the similarity between cluster models

Suppose in a 5-agent network, datasets owned by the first three agents A0, A1 and A2 are sampled from one pattern and the rest are sampled from another pattern. Figure 5.1 shows the raw data point for those agents. Based on the raw data points, it is a challenge to decide algorithmically which agents are similar

(and so belong to the same pattern) and which are different (and so belong to different patterns).

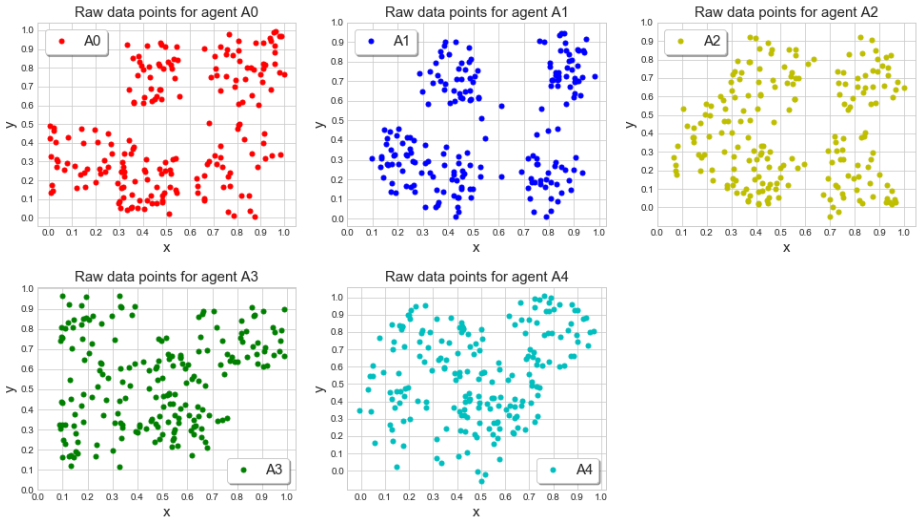


Figure 5.1: Raw data points for each agent

The standard output of a cluster model is a set of centroids. Here we assume that each agent is fixed to have 5 clusters. Figure 5.2 shows the corresponding five centroids found by K-means. The centroids found by different agents are separated by different colour and shape. The widely used L1 and L2-norms could be used to measure the similarity between agents. Obviously, centroids of agent A0, A1 and A2 are similar, but differ from agent A3 and A4. However, this method ignores relevant information about the clusters. Each cluster is represented as centroid only, and the shape of cluster and the size of cluster are not considered.

From the perspective of data points, each agent is representing a distribution of data. Standard EMD is one way to compare distributions, in which the cost of moving each data point from one distribution to another is computed. But it is expensive. This motivates us to find a new method to measure the similarity.

In this section, we propose a method to measure the similarity between the cluster models inferred by different agents. This similarity is further used to estimate the number of patterns and identify the patterns. Note that we focus on the specific scenario that agents know the size of network (S1) and must finish with identical models (A2). We will select a measure based on the results of this scenario, and then transfer it for using in other scenarios. We consider two EMD-based similarity measurements: EMD between centroids weighted by the size of corresponding cluster (Weighted EMD) and EMD between all raw data points of

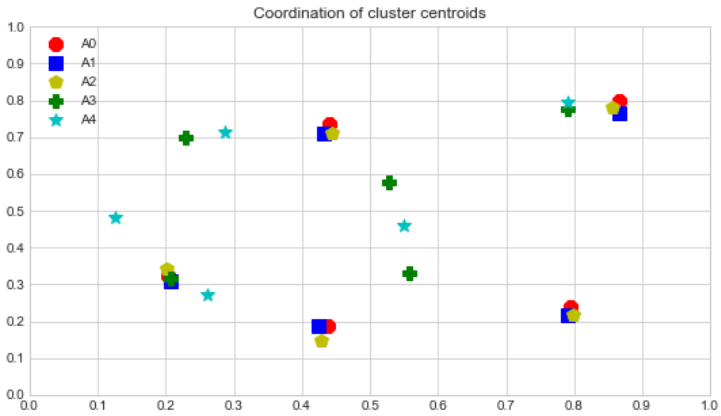


Figure 5.2: Centroids for each agent

agents (Standard EMD). To evaluate the performance of EMD based methods, we compare the widely used L1-norm Euclidean distance and L2-norm Euclidean distance with Standard EMD and Weighted EMD. The four different methods to measure the pair-wise similarity between agents are listed as follows:

1. **L1-norm distance**. It measures the sum of euclidean distance between centroids.

2. **L2-norm distance**. The L2-norm of euclidean distance between centroids is computed. By this method, the overall distance is treated as an n-dimensional vector, where each vector is computed by distance between centroids.

3. **Weighted EMD**. We assume that agents are denoted by a set of centroids. The similarity between agents are represented by EMD distance between the centroids. Take the 5 agents in figure 5.2 as an example, the similarity between agent A_1 and A_2 are computed as the EMD between the 5 red circular centroids and 5 blue rectangular centroids.

4. **Standard EMD**. We assume that agents are denoted by all raw data points instead. The similarity between two agents is computed as the EMD between all points of those agents.

Figure 5.3 shows the hierarchical clustering result based on the four different methods [1]. The horizontal axis of the dendrogram represents the objects and clusters and the vertical axis represents the distance between clusters. The hor-

[1] We use the measures to create distance tables, and then apply standard hierarchical clustering on that distance table.

izontal bar between the two clusters is labelled with the distance (dissimilarity) of these two clusters. When Euclidean distance is used to measure the similarity, agent A_0, A_1 and A_2 are close to each other, but agent A_3 and A_4 are far away from each other (10 times larger than the average distance of first three agents). However, when EMD is used, the first three agents stay close with each other and A_3 and A_4 also cluster together. The top two figures are more likely to estimate that there are three different groups since A_3 and A_4 are far away from each other.

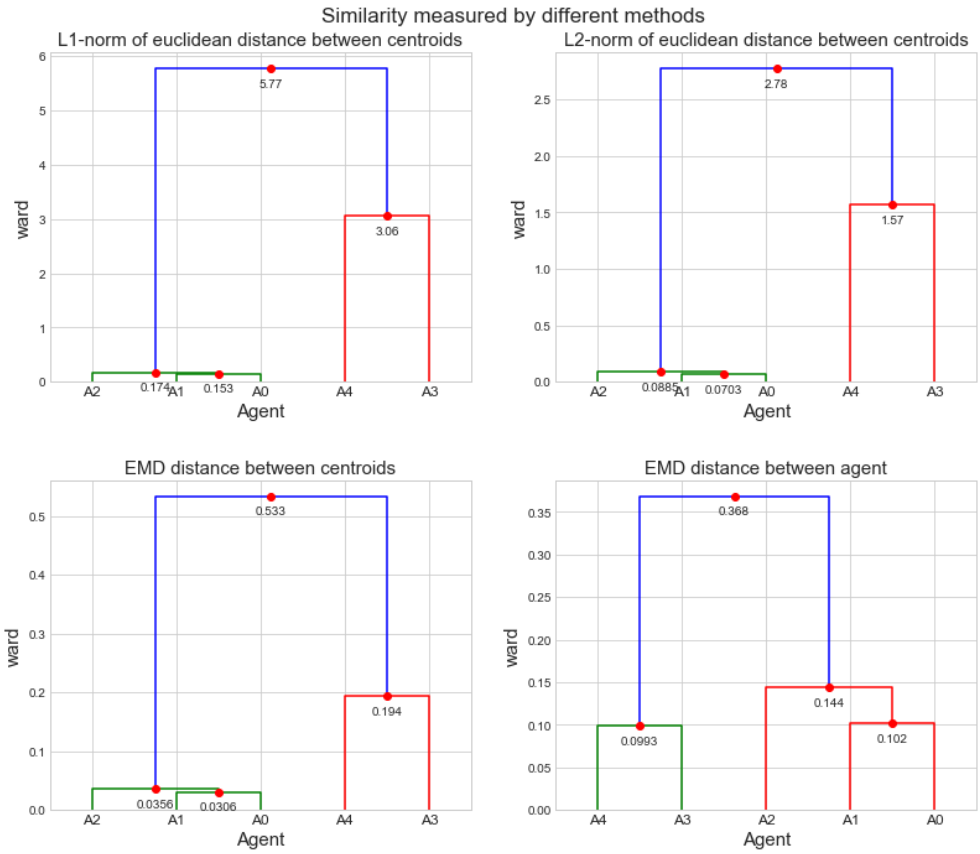


Figure 5.3: Hierarchical clustering by different methods

Since the hierarchical clustering result is available, now the question is how to select a cut-off value that splits the dendrogram into subgroups such that agents in the same group are similar but agents in different groups are different. Although there are some common methods, including the elbow method, silhouette method, cross-validation and information-theoretic approach, there is no standard method to pick the cut-off points. There are two popular methods in the literature: inconsistency method and elbow method. Both methods aim to find the biggest increment in distance during the merging process. Taking EMD between agents in figure 5.3 as an example, the dissimilarity between the new cluster C_1 (merged

by A4 and A3) and new cluster C_2 (merged by A_0, A_1 and A_2) is 0.368, more than 2 times larger than its last merge 0.144. It suggests that C_1 and C_2 are dissimilar and they should not merge into a new cluster.

In a hierarchical clustering tree, one way to determine the natural cluster divisions to compare the height of each link with the heights of neighbouring links below it. If there are no distinct divisions between objects joined at this level, the height of these two links is approximately the same. On the other hand, a link is considered to be inconsistent with another link if the heights differ noticeably from each other. The inconsistency method compares each cluster's merge height h to the average and normalises it by the standard deviation std formed over the previous h levels:

$$inconsistency = \frac{h - avg}{std} \tag{5.1}$$

where the inconsistency matrix consist of average, std, count and inconsistency for each merge. This method finds the biggest increment during the merging steps, and a cut-off value is automatically confirmed at the merging step with the biggest increment. The sub-figure in the middle of figure 5.7 shows the merging steps, where h is set to be 3. The biggest increment is detected in the final merge step, where new cluster C_1 (merged by A_4 and A_3) is merged with new cluster C_2 (merged by A_0, A_1 and A_2). Based on this estimation, the estimated number of groups P is 2: $[A_4, A_3]$ and $[A_0, A_1, A_2]$.

However, this inconsistency method heavily depends on the depth of the tree that the average is calculated over. If the default depth is assumed to be 2, it assumes that only the previous distance matters. Moreover, the previous d levels' heights aren't normally distributed, but increase monotonically. That is because the hierarchical clustering method always merges the closest two clusters (agents). Simply treating the current level as an outlier of a normal distribution is not right, as it is expected to be bigger. In this section, we use the elbow method to estimate the number of patterns.

The elbow method aims to find the merging step where there is a huge acceleration of distance growth. Algorithm 18 shows the main steps to estimate the cut-off value by the Elbow method. The difference of a new merge is measured against the last merge. If the difference is maximised at a merge step, it indicates the cut-off value should be the last distance before merging. The sub-figure in the right side of figure 5.7 shows the increment during merging steps. The biggest increment is detected in the final merge step, so there are two groups $[A_3, A_4]$ and $[A_0, A_1, A_2]$.

Algorithm 18: Elbow method to estimate the cut-off value

Input: Distance metrics during the merge process D $=\{d_1, d_2, \cdots, d_m\}$;
Output: The cut-off value r;
Difference metric Out=[];
for $i = 1; i \leq m$ **do**
    Out[i] = d[i+1] - d[i] ;
Obtain the index ix of the maximum value in metric Out;
r = D[ix];

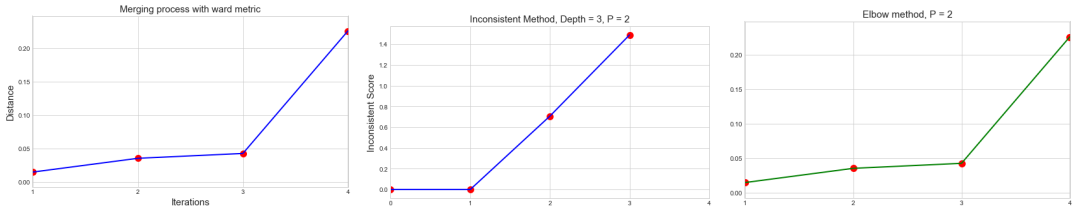


Figure 5.7: Estimation process by inconsistent and elbow method

5.2.1 Standard EMD based method for similarity measurement

Based on the similarity measurement, we are interested in how accurately the algorithms put the data points into the clusters and predict the number of patterns compared to the ground truth. The accuracy against data points and against ground truth are the same measure used in the previous chapters. The experiment is carried out in a 10-agent network, where the first 7 agents are from one pattern and the rest are from the another pattern and it is repeated for 50 runs.

Note that in the experiment we assume that agents know the size of network (S1) and must finish with identical model (A2). The overall performance of the distributed algorithms, which embed the four similarity measurements as distance metrics, are measured. Figure 19 shows the algorithm used for evaluating the similarity measures.

Figure 5.8 shows the accuracy on data points against ground truth. It shows that the accuracy achieved when the algorithm uses either of the two EMD methods is greater than when using L1-norm and L2-norm distance.

Figure 5.9 shows the number of patterns estimated by different methods. Over the 50 runs, algorithms using EMD between centroids and EMD between agents predicted the same number of patterns as ground truth. Although the method using L1-norm distance predicts the number of patterns correctly 42 times, the inferred number of patterns could be as high as 5.

Algorithm 19: Adaptive asynchronous distributed clustering algorithm for multiple patterns

```
Agents run local clustering to completion ;
Agents transmit summaries with agent ID to neighbours;
while not stabilised or messages received do
    if Receive terminate message then
        Update local knowledge (number of patterns, pattern assignment);
        Transmit terminate message to all neighbours;
        Terminated;
    else if Receive final summary then
        Compare the similarity between agents and infer the patterns (based
         on some distance metrics);
        Compute global view for each pattern;
        Attach a timestamp to this produced model;
        Send converge command to neighbours;
    else
        if Have not seen this message before then
            Restore the summary received;
            Transmit it;
```

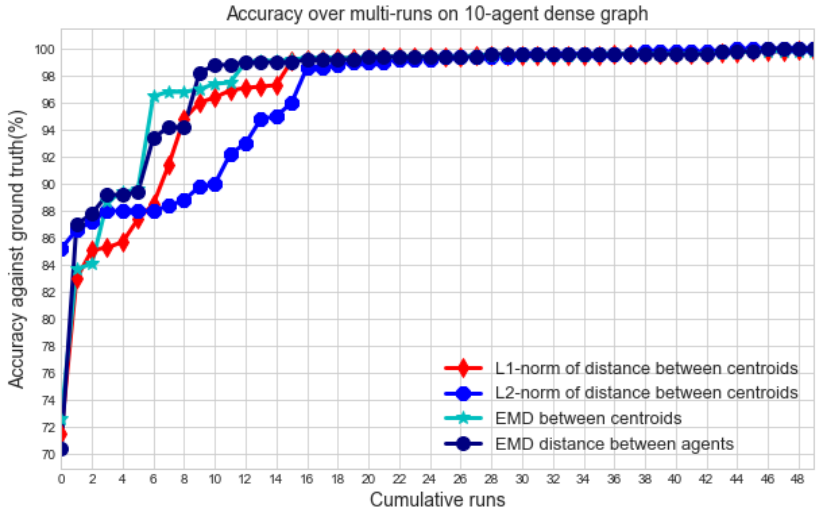


Figure 5.8: Accuracy on data points against ground truth with different similarity metrics

To summarise, an EMD based method is a potential way to represent the similarity. This method does not assume that the number of clusters or data points for different agents are necessarily consistent. The previous chapter considered cases where we do not assume the number of clusters before we start. If we now accept that agents may be receiving data from different distributions (e.g. different environmental conditions being reported by sensors in different locations),

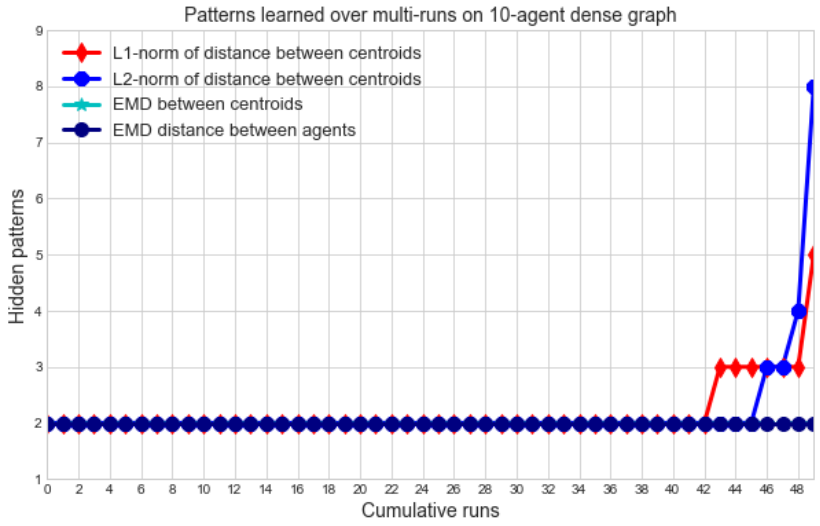


Figure 5.9: The number of estimated patterns with different similarity metrics

then it is also possible that there may be different numbers of clusters in each distribution. If we relax the assumption that distributions have the same number of clusters, then the standard versions of the L1- and L2-norm can no longer be used.

5.2.2 Optimised EMD based method for similarity measurement

In many practical applications, the number of cluster for different distributions are not necessarily the same. In that case, L1- and L2-norm can no longer be used. This motivates us to find a similarity measurement that works even when the number of clusters are different. We propose a reversible EMD between points and centroids to measure the similarity between agents. It computes the EMD between the centroids of one agent and raw points of another agent, and vice versa. The reversible EMD between points and centroids is defined as the average distance of these two one-way EMD (refer to figure 5.10). We now evaluate the performance of standard EMD, weighted EMD and reversible EMD, and compare them to the state of the art Wavelet EMD [Shirdhonkar and Jacobs, 2008] and Robust EMD [Pele and Werman, 2009].

Consider two agents A_1 and A_2. Each agent is denoted by some features, including centroids C_i, counts η_i, and regenerated dataset Γ_i from summary description received. Table 5.1 shows the description of the different methods compared in the experiment.

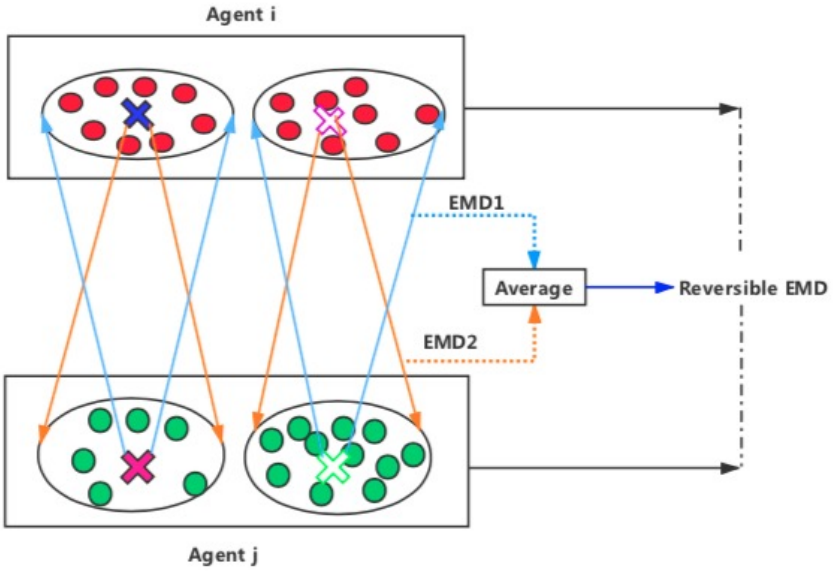


Figure 5.10: Reversible EMD between agents

Table 5.1: Description of different methods

Name	Descriptions
Weighted EMD	Compute the EMD between centroids C_1 of agent A_1 and C_2 of agent A_2
Reversible EMD	Compute the average of EMD between A_1's centroids C_1 to A_2's sampled points Γ_2, and the reverse
Standard EMD	Compute the EMD between sampled dataset Γ_1 of agent A_1 and Γ_2 of agent A_2
Wavelet EMD	Compute the Wavelet EMD between regenerated dataset Γ_1 of agent A_1 and Γ_2 of agent A_2
Robust EMD	Compute the Robust EMD between regenerated dataset Γ_1 of agent A_1 and Γ_2 of agent A_2

To evaluate these measures, we again run Algorithm 19 with each EMD measure embedded inside, each agent starts after a random delay (uniformly in [0,0.3]) and the delay for sending a message is random selected from [0.5,1]. Note that in the experiment agents know the size of network (S1) and agents must finish with identical model (A2). Each agent is fixed to have 5 clusters. The experiment is carried out in a 10-agent network and repeated for 50 runs, and two splits of agents are tested: 7:3 and 6:4. The basic setting is as follows:

1. **Dataset.** Three different synthetic datasets, that generated from fully symmetric multi-dimensional Gaussians, partially symmetric multi-dimensional Gaussians, and uniform distributions, are used to test the algorithm.

2. **Measurement**. Three different accuracies are measured:

2.1. Accuracy against centralised method, denoted by Acc_c.

2.2. Accuracy against the ground truth, denoted by Acc_g.

2.3. Accuracy against patterns, denoted by Acc_p, which measures how accurately the agents are put into the same pattern as ground truth. Since in the current scenario and algorithm all agents finish with the same global model, we simply compute the pairwise accuracy against ground truth.

2.4. Convergence time.

The measurements used in previous chapter only consider the case of True Positive, where only data points that are assigned to the same cluster as centralised method or ground truth are counted. Here, we also consider the Rand index method to calculate the accuracy in a pair-wise way. The Rand index computes the percentage of correct decisions made by the algorithm compared to the benchmark classifications as the following formula [Rand, 1971]:

$$RI = \frac{TP + TN}{TP + FP + FN + TN} \tag{5.2}$$

where TP is the number of true positives, TN is the number of true negatives, FP is the number of false positives, and FN is the number of false negatives. Note that RI assumes that TP and TN are equally important. We also consider a weighted version of rand index WRI:

$$WRI = \frac{TP + \delta TN}{TP + FN + \delta FP + \delta TN} \tag{5.3}$$

where the $\delta \geq 0$ is the weight applied to pairs that should be negative. Also, we want to balance the contribution of false negatives by weighting recall through parameter δ, and the F-measure is used, and calculated by the following formula [Sasaki et al., 2007]:

$$F = \frac{(\delta^2 + 1) * P * R}{\delta^2 * P + R} \tag{5.4}$$

where precision P is defined as $P = \frac{TP}{TP+FP}$ and recall R is denoted by $R = \frac{TP}{TP+FN}$. Note that how agents are split affects the parameter δ, and we assume that δ is defined as the ratio of size of two patterns.

We only show the results when K-means is used as clustering algorithm, and

the results on GMM and SG are attached in appendix 8.3, where the results are similar. Figure 5.11 shows the accuracy against centralised method when the underlying datasets are generated from fully symmetric Gaussians. EMD achieves the highest accuracy against centralised K-means in all four measures, followed by weighted EMD between centroids. But the difference is small (refer to the actual result in table 5.2). Of all five methods, the accuracy obtained by wavelet EMD is the worst. However, all measures are above 90%, which indicates that all methods perform well when underlying datasets are generated from fully symmetric Gaussians.

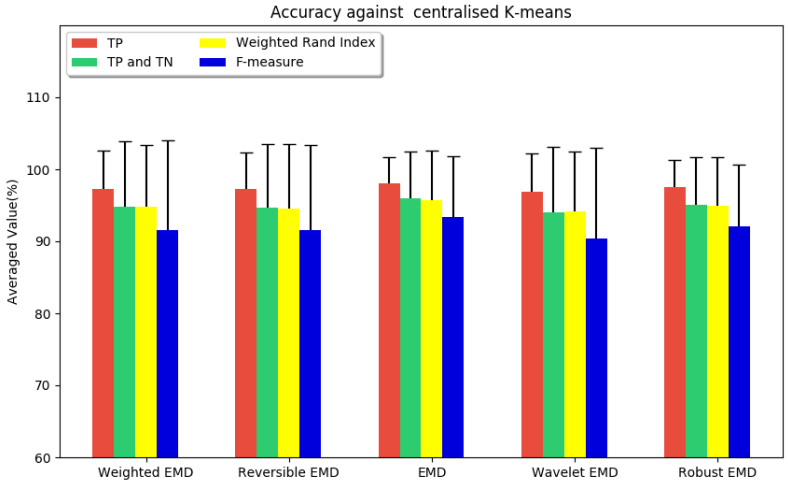


Figure 5.11: Accuracy measures against centralised k-means with symmetric Gaussians

Table 5.2: Accuracy measures against centralised k-means

Name	TP	TP and TN	WRI	F-measure
Weighted EMD	97.28(5.33)	94.77(9.14)	94.78(8.59)	91.55(12.52)
Reversible EMD	97.26(5.08)	94.7(8.82)	94.52(9.00)	91.59(11.73)
Standard EMD	98.02(3.73)	96.02(6.49)	95.77(6.85)	93.43(8.38)
Wavelet EMD	96.9(5.27)	94.09(9.02)	94.22(8.31)	90.42(12.51)
Robust EMD	97.5(3.81)	95.11(6.58)	94.91(6.84)	92.12(8.49)

Compared to the accuracy against centralised K-means, all measures on accuracy against ground truth dropped (see figure 5.12). Note that there are little difference in accuracy achieved by weighted EMD, Reversible EMD and EMD. But still, these accuracies are higher than wavelet EMD and robust EMD.

Although there is little difference in accuracy against centralised method and ground truth between various methods, we expect that accuracy against pattern

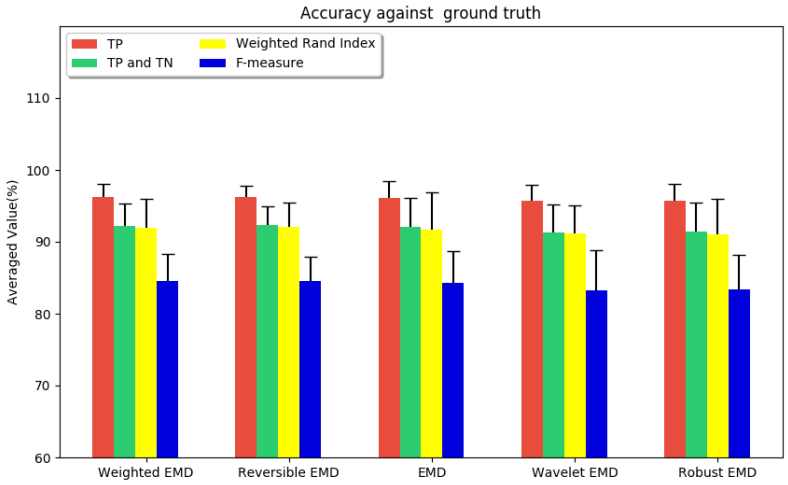


Figure 5.12: Accuracy against ground truth with symmetric Gaussians

will vary with the distance methods. That is because similarity between agents is critical for assigning agents to different groups. The distance method that describe the similarity between agents well is more likely to predict the same agent assignment as ground truth patterns. Figure 5.13 shows how accurately the algorithm puts the agents into the correct groups. This time, weighted EMD outperforms other methods, followed by EMD. Again, wavelet EMD performs the worst.

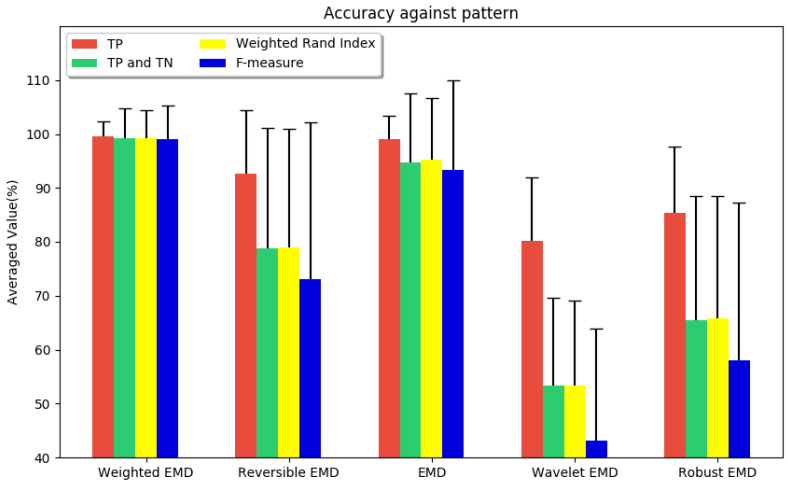


Figure 5.13: Accuracy against pattern with symmetric Gaussians

Figure 5.14, 5.15 and 5.16 show the accuracy against centralised K-means, ground truth and pattern separately when data points are generated from partially sym-

metric Gaussians. In terms of accuracy against centralised K-means, EMD is slightly higher than other methods, followed by weighted EMD. EMD achieves the highest accuracy against ground truth. Again, there is little difference in accuracy against centralised K-means and ground truth between various methods. When accuracy against pattern is measured, weighted EMD outperforms other methods, followed by EMD. Again, wavelet EMD performs worst. Note that the F-measure is low for Reversible EMD, Wavelet EMD and Robust EMD. That is because those methods separate similar agents into different sub-patterns, and the F-measure penalise those methods.

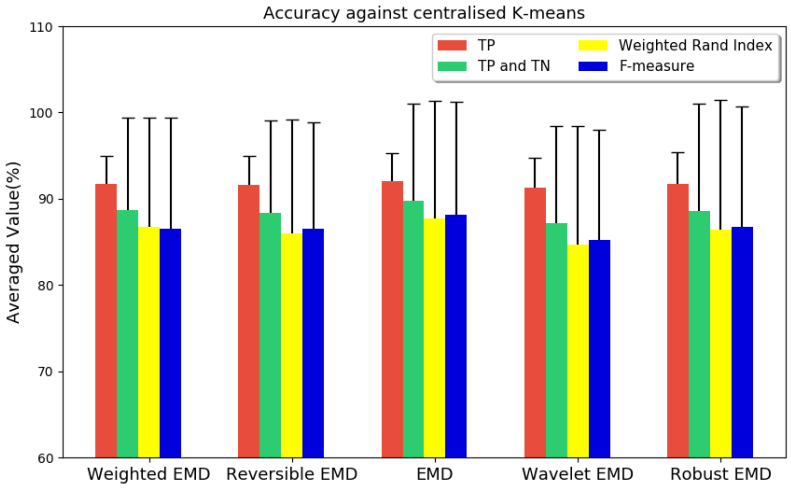


Figure 5.14: Accuracy against centralised k-means with asymmetric Gaussians

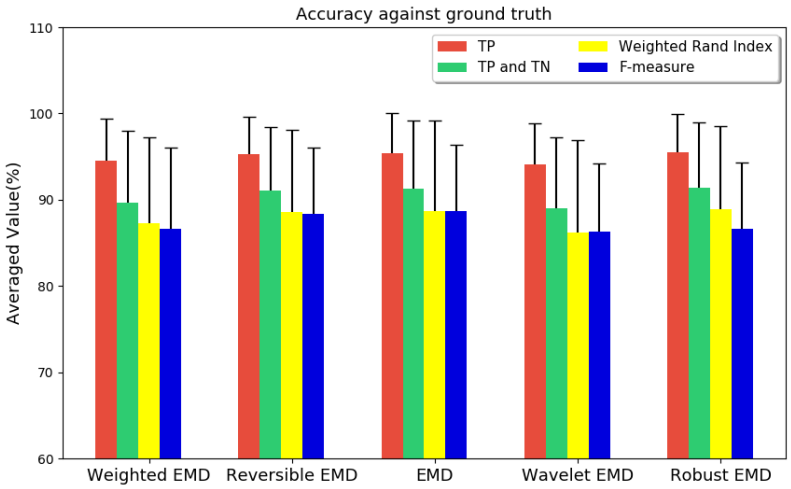


Figure 5.15: Accuracy against ground truth with asymmetric Gaussians

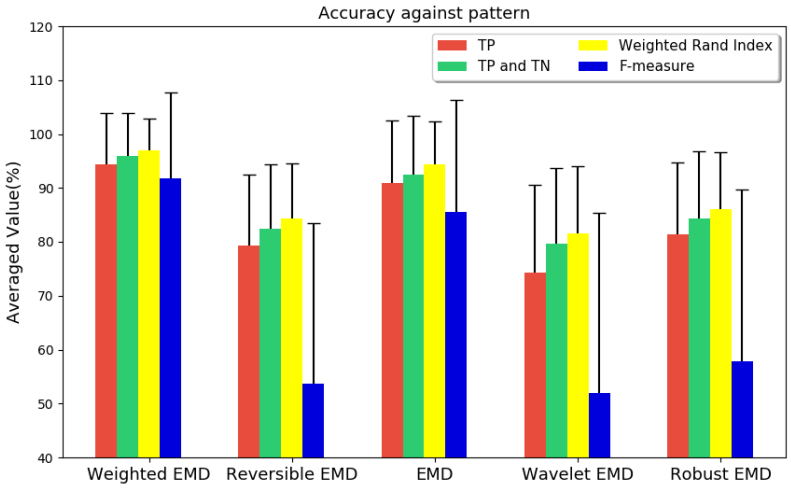


Figure 5.16: Accuracy against pattern with asymmetric Gaussians

When the underlying datasets are generated from uniform distributions, it is a challenge for K-means to cluster. Again, EMD achieves slightly higher accuracy against centralised K-means and ground truth than other methods (see 5.17 and 5.18), and there is again little difference between all methods. When accuracy against pattern is measured, weighted EMD outperforms other methods, followed by EMD. Wavelet EMD and robust EMD perform worst in accuracy against patterns.

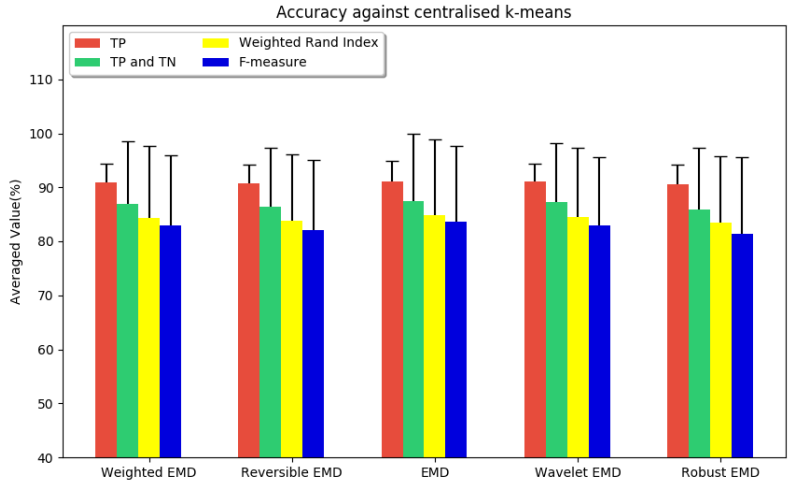


Figure 5.17: Accuracy against centralised k-means with uniformly distributed dataset

In addition to the measurement above, we also consider the computation cost

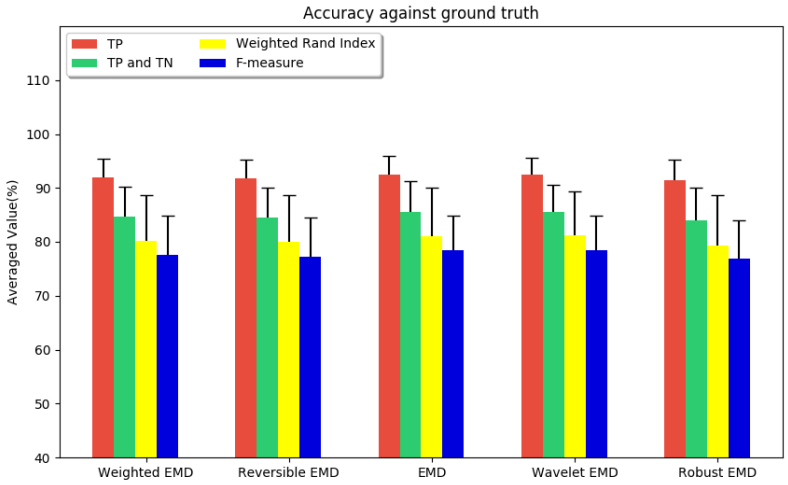


Figure 5.18: Accuracy against ground truth with uniformly distributed dataset

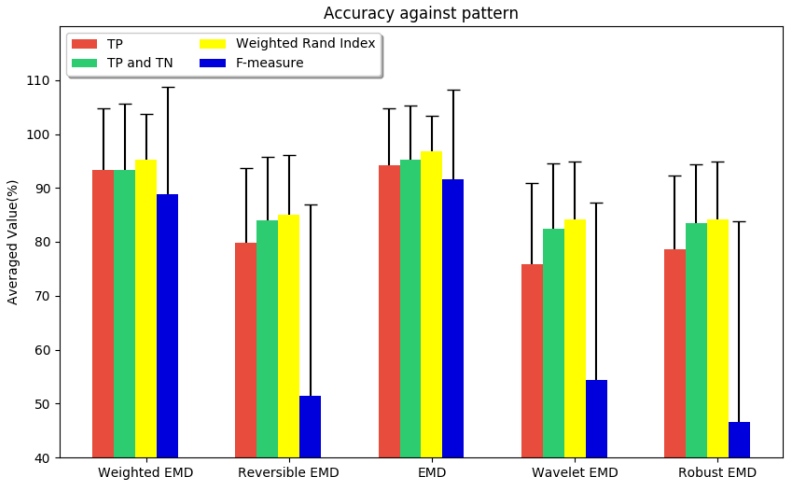


Figure 5.19: Accuracy against pattern with uniformly distributed dataset

for weighted EMD, standard EMD, Wavelet EMD and Robust EMD. The computation cost is measured by the convergence time, which measures the time interval between the first agent initialising its work and the last agent finishing the process.

Figure 5.20 shows the convergence time over 50 runs. It shows that wavelet EMD converges fast, followed by weighted EMD. Robust EMD requires the longest time to converge. The reason why the robust EMD is the slowest lies in the set-up time. The wavelet EMD is the fastest, but as shown above, it performs worse than EMD and weighted EMD in term of accuracy against patterns.

Overall, weighted EMD performs similar in term of accuracy against centralised method, ground truth and patterns, but converges faster than EMD. That is because weighted EMD only computes the work of moving centroids while EMD has to compute the work of moving all raw data points.

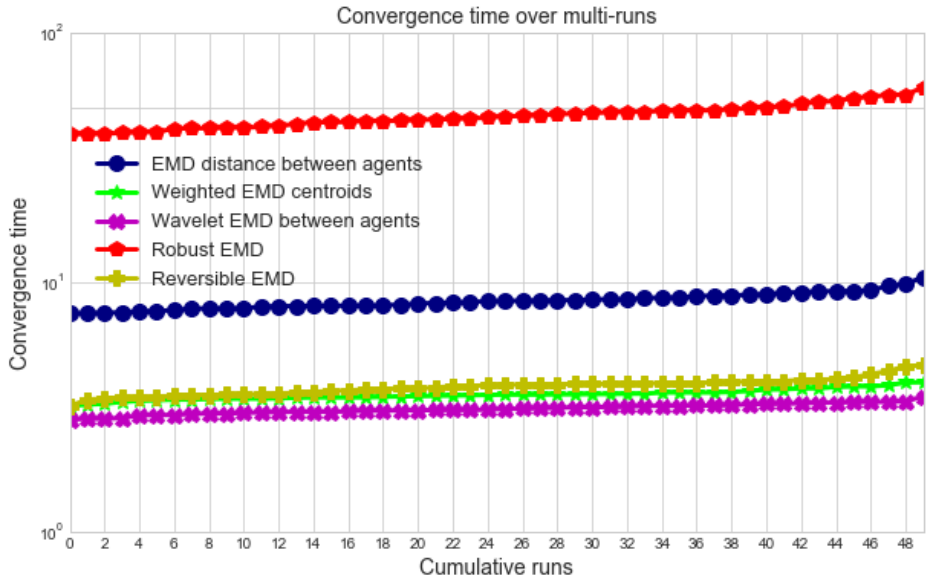


Figure 5.20: Comparison of convergence time

The experiments shows that the proposed weighted EMD is a potential way to measure the similarity between agents. It performs better in accuracy on data point and estimated number of patterns than Euclidean distance L1-norm and L2-norm. Similar to standard EMD, it performs well in term of accuracy against centralised method, ground truth and pattern, but converges much faster. Compared to the state-of-arts, it converges faster than the robust EMD, and slightly slower than Wavelet EMD. However, it achieves higher accuracy than robust EMD and wavelet EMD. Since Weighted EMD performs better than Reversible EMD, we will use Weighted EMD as distance metric to measure the similarity between agents hereafter.

5.3 Sharing strategy for different scenarios

From now on, we will adopt that similarity measure and apply to the other scenarios. In this section, we go back the general question of how to understand the environment if there are multiple patterns, and we will use the selected Weighted EMD as the distance measure. On top of the two scenarios (S1 and S2: agents know the size of network or agents are only aware of their neighbours) and two

different assumptions (A1 and A2: final models may be different, or must be identical) considered in previous chapters, we also consider two different potential requirements in the network. Suppose that a centralising agent A_i produces the patterns and models for each sub-pattern after seeing all cluster models, and it knows which particular sub-pattern each agent belongs to. After that, the centralising agent A_i shares the patterns and cluster models for each sub-pattern with its neighbours. There is then a question over how much information on each individual agent to share with other agents. To respect privacy, it may restrict shared information to just the number of patterns and corresponding summary description of each sub-pattern, or it could also tell its neighbours the details of which particular sub-pattern every agent belongs to. So two different requirements are assumed at the start:

R_1: Each agent is told the number of patterns and summary description of each pattern, but not the individual agent assignments to the patterns.

R_2: Each agent is told the number of patterns, the pattern description and the details of assignment of all agents.

Although $R1$ preserves some privacy for the agents, the disadvantage is that agents have to infer which sub-pattern they belong to based on their own computation. Note that this computation may introduce errors that were not present in the original inferred model that was communicated to the network. If $R2$ is applied, agents are told their assignment, but its assignment is also known to other agents.

Figure 5.21 shows the different scenarios considered in this chapter. For the variants which push raw data points or a basic model to a central agent, we assume that each agent is told the patterns and the group assignment of all agents.

Next, we describe the sharing strategies and process steps for the different scenarios considered in this chapter. First, we consider the distributed methods. If agents may finish with different models (A1), any agent that receives information from all other agents before it receives a global model should compute a global model. If it still has not received a global model from any of its neighbours, it then transmits a description of the model it has discovered, including for each pattern a description of the cluster centroids, count, shape and density. An agent may receive multiple global patterns, in which case it should keep the first model it received (or produced by itself), and it should only relay that model to its neigh-

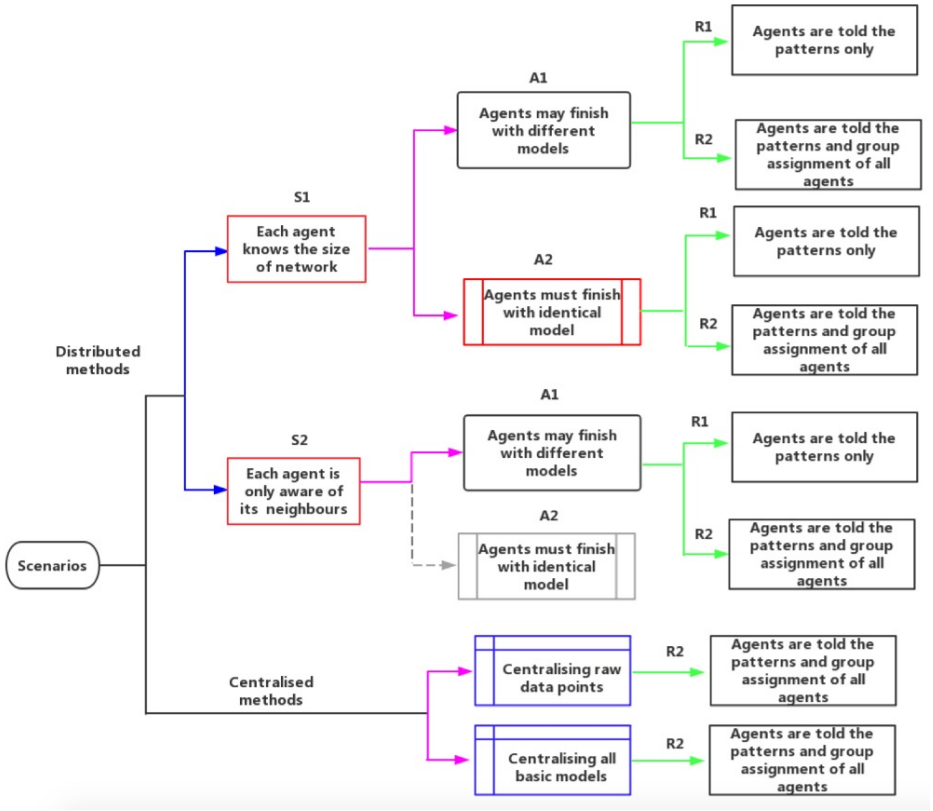


Figure 5.21: Different scenarios considered for multiple patterns

bours. When agents must finish with identical models (A2), each agent chooses the model with the earliest timestamp as its final model. Any agent learned or received a global model, should relay this model to its neighbours.

If agents are not told the group assignment of all agents (R1), each agent must infer which sub-pattern it belongs to by:

V_1: comparing its basic model with patterns received. Weighted EMD is used to measure the difference between agent and sub-patterns. Each agent chooses the pattern with the smallest distance as its final assignment.

V_2: comparing its raw data with patterns received. Similarly, Standard EMD between raw data points and sub-patterns is used to measure the difference. Each agent chooses the pattern with smallest distance as its final assignment.

When agents know the group assignment of all agents (R2), both the basic description for each pattern and the assignment of agents to patterns must be shared. In this case, each agent simply reads its assignment.

For centralised methods, since a tree-based structure is applied and information

flows up to the root node, we assume that the centralising agent knows the size of all network (S1). Also, the central agent is the only agent that produces a final model, so each agent will converge with the same model (A2). After central agent learns the global model, a second communication is initialised to broadcast the model to all its child nodes. Thus all agents are told the number of patterns and group assignment of all agents (R2).

5.4 Measurement

Measuring the accuracy against data points and ground truth is not enough when there are multiple patterns in the network. Since now the aim of algorithm is to assign agents to groups and to infer the data distribution for each group, we need an additional measurement accuracy against pattern, which measures how accurately the agents are assigned into groups or patterns compared to the original ground truth. Since two new requirements (R1 and R2) are introduced, accuracy against pattern under these new requirement is introduced.

5.4.1 Accuracy against patterns with known assignment (R2)

Accuracy against pattern is an important quota to measure how accurately the algorithm puts agent into the right groups against ground truth patterns. Note that predicting the right number of patterns only is not enough. It is possible that agents are put into the wrong sub-patterns, but the right number of patterns has been preserved. When agent knows the details of assignment of all agents (R2), we can compute the accuracy in a pair-wise way. The accuracy against pattern is computed as:

$$Acc_p = \frac{\sum_{i=1}^{n}({Acc_p}^i)}{n} \tag{5.5}$$

Where ${Acc_p}^i$ denotes the estimation by agent i. For each agent i, it computes the estimation by comparing the inferred pattern with ground truth pattern directly, and it could be computed by the following two methods:

Me_1: For agent i and other agents in the same inferred group, are they assigned in the same group as ground truth? Count the number of pairs $\{i, j\}$ ($j \in G_i$ and $j! = i$) that are assigned in the same group as ground truth [2], or

[2] G_i denotes the a sub-group where agent i is in.

Me_2: For all agents in the learned pattern, are they assigned in the same group as ground truth? Count the number of pairs $\{i, j\}$ in sub-group G_m ($j \in G_m$, $i \in G_m$ and $j! = i$) that are in the same sub-group as ground truth.

The difference between these two methods is that Me_1 only considers an agent and its neighbours that in agent's group while all pairwise agents in the learned patterns are considered in Me_2. Suppose the pattern learned by A_0 is $[[A_0, A_1, A_2], [A_3, A_4]]$. All pairwise agents in the first sub-group are considered: $[A_0, A_1]$, $[A_0, A_2]$ and $[A_1, A_2]$ if Me_1 method is used. If Me_2 is used instead, then all possible pairwise agents are considered: $[A_0, A_1]$, $[A_0, A_2]$, $[A_1, A_2]$ and $[A_3, A_4]$. The pair $[A_3, A_4]$ is not considered in measurement Me_1 but is taken into account in measurement Me_2.

5.4.2 Accuracy against patterns without known assignment (R1)

When each agent only knows the number of patterns and which sub-pattern it belongs to (R1), how should we measure the accuracy? It is a challenge since the prediction of the assignment for agents may be based on different models. Suppose that in a 5-agent network $(A_i, A_j, A_k, A_l, A_n)$, agent A_i and A_j produced final models (m_i and m_j) at some timestamp t_r. Then these two different models are broadcasted to the other agents.

When agents must finish with identical model, the centralising agent A_i may have put agent A_j and A_k into the same group, but agent A_k has put itself into a different group. The accuracy measurement depends on which agent's view is used. Things are even worse when agents may finish with different models. Agent A_k receives model m_i, which puts agent A_i, A_j and A_k into a group G_1, and agent A_k puts itself in that group. Agent A_l receives model m_j, which had a group G_2 contains A_i, A_l and A_n, and A_l puts itself in that group. Now the accuracy measurement depends on all centralising agents' view. In this section, we propose three different potential solutions to measure the accuracy against pattern.

The first one is to build a symmetric table based on agents' prediction. It could be used in scenario with identical model or different models. Based on the prediction, agents who make the same prediction are grouped together. Figure 5.22 shows one example to build the symmetric matrix in a 5-agent network. Agent A_4 produced the final model and broadcast its model to the rest of agents. Then

each agent makes a prediction based on the model received. Agent A_4 produced the model, so it knows which sub-pattern it is in. Agent A_2, A_3 and A_5 predict they are in sub-pattern P_2, so they are grouped together. Agent A_1 predict it is in sub-pattern P_1, so it is alone. Thus, the final table is :

$$[[A_1], [A_2, A_3, A_5], [A_4]]$$

Then, this table is compared to ground truth patterns:

$$[[A_1], [A_3, A4], [A_2, A_5]]$$

By this method, the common knowledge of predictions is considered, and agents who make the same prediction are grouped together.

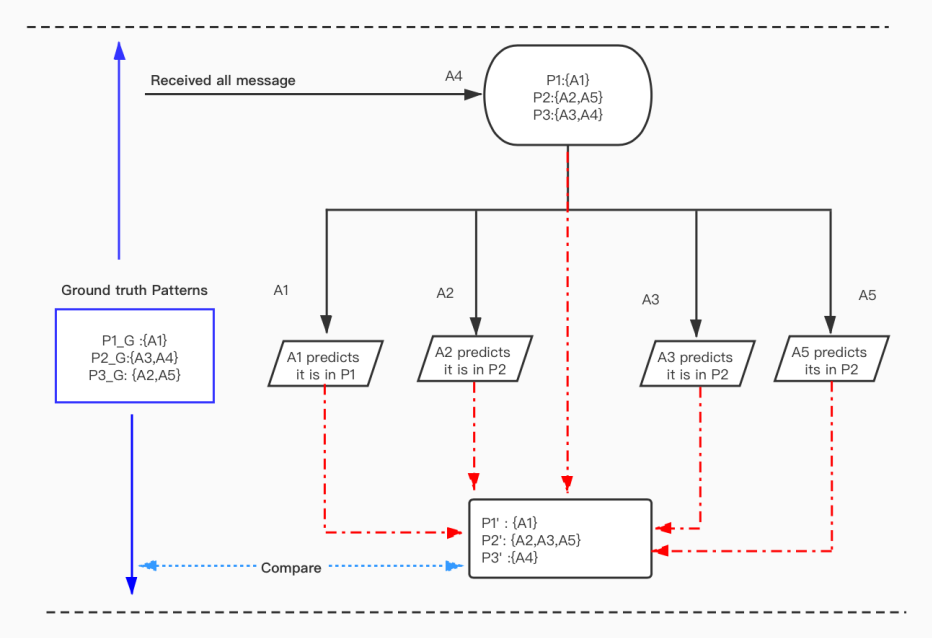


Figure 5.22: Symmetric table built by agents' prediction

The first method is not working if it is further extended to the case when there are multiple different models in the network, and it inspires us to propose a new method. The second method is to build an asymmetric table (denoted by MD1). Algorithm 20 shows the algorithm to build an asymmetric table. First, the actual sub-pattern s of patterns computed by centralising agent which this agent i belongs to is obtained. Then this index s is compared to the prediction p. If agent i made a correct prediction ($p = s$), then agents that together with and apart from this agent within model M are stored. However, agent i may make the wrong prediction. In this case, agent i is inserted to the p^{th} sub-pattern of

model M. Actually, agent i is put in another sub-pattern $\bar{p}$ by the centralising agent. So agent i is removed from sub-pattern $\bar{p}$. Again, the agents that together with and apart from agent i are stored. Then the asymmetric table is built when all agent find out who they are together with and apart from.

Algorithm 20: Method to estimate the relationship with other agents for individual agent

```
1  Input: Prediction p, Model received M;
2  Output: T_s: set of agents together with agent i, T_a: set of agents apart
     from agent i ;
3  Find the index s of agent i in the received Model M;
4  if s == p then
5      Add agents that together with agent i to T_s ;
6      Add agents that apart from agent i to T_a;
7  else
8      Insert agent i in the pth pattern of model M;
9      Remove agent i in other patterns (not in pth pattern) of model M;
10     Add agents that together with agent i to T_s ;
11     Add agents that apart from agent i to T_a;
```

Figure 5.23 shows one example to build an asymmetric table. Agent A_2 and A_4 computed the final mode m_2 and m_4. Then model m_2 is sent to agent A_1 and m_4 is sent to agent A_3 and agent A_5. Agent A_1 predicts that it is in the sub-pattern P1. Based on the centralised agent A_2, the prediction is right and the agents that together with and apart from A_1 should be consistent with that in model m_2 (together with $[A_3]$ and apart from $[A_2, A_4, A_5]$). However, agent A_3 made the wrong prediction: it predicts it is in second sub-pattern, but actually it is in the third sub-pattern of model m_4. By inserting agent A_3 to the predicted sub-pattern and deleting the actual location of agent A_3, now agent A_3 is assumed to together with $[A_2, A_5]$ and apart from $[A_1, A_4]$, which differs significantly from the model m_4.

By this method, the correlation between agents is ignored. Two agents that made the same prediction based on same models may have different estimations. However, different from the symmetric table, both agent's prediction and the centralised method are considered. It is suitable for the case that there are multiple different patterns in the network.

The last method compares the patterns learned by centralised agent with the ground truth directly (denoted by MD2). Take agents in figure 5.23 as an example, agents A_2 and A_4 learned the global patterns $[[A_1, A_3], [A_4], [A_2, A_5]]$ and

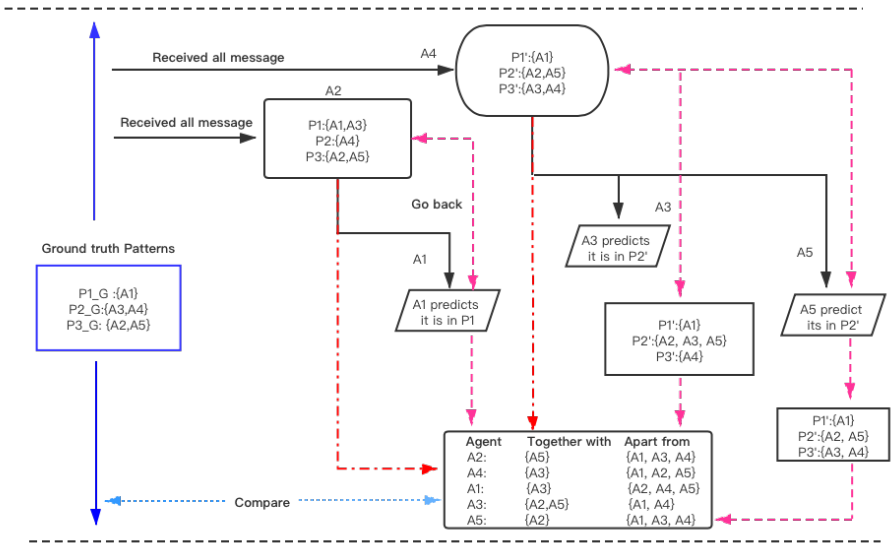


Figure 5.23: Asymmetric table built by agents' prediction

$[[A_1], [A_2, A_5], [A_3, A_4]]$. So we compare them with the ground truth:

$$[[A_1], [A_3, A_4], [A_2, A_5]]$$

By this method, the errors introduced by prediction by agents have been avoided. To evaluate the how accurately the algorithm puts agents into the same group as ground truth, method MD1 and MD2 are used.

5.5 Time Complexity

In this section, we analyse the time complexity. Compared to the proposed algorithms in last chapter, there are two new steps: computing the similarity between different cluster models and doing spectral clustering. Given N data points, computing the similarity (Earth Mover's Distance) takes time in $\mathcal{O}(N^2)$ and the time complexity of doing spectral clustering is bounded by $\mathcal{O}(N^3)$.

In last chapter, we have shown that, no matter whether the size of network is known or not, the time complexity for the three distributed clustering algorithm variants is bounded by $\mathcal{O}(N^2)$, and the time complexity for the two centralised methods is also bounded by $\mathcal{O}(N^2)$. We can conclude that the time complexity for distributed algorithms and centralised algorithms is bounded by $\mathcal{O}(N^3)$.

5.6 Empirical Evaluation

In this section, we show the experimental result when K-means, GMM and SG are used as the clustering algorithm on a 10-agent sparse or dense graph. Each node is given 200 data points and the ground truth number of clusters in all simulations is 5. Each message transmission delay is generated uniformly from the range [0.5, 1.0], and nodes begin their work at time randomly selected in [0, 0.1s]. Three different underlying 2-dimensional datasets are tested: fully symmetric Gaussians, partially symmetric Gaussians and uniformly distributed datasets. Four different splits of agents are tested: three with two patterns (5:5, 7:3, 9:1) and one with three patterns (5:3:2). In addition to convergence time, the number of transmitted message, number of clusters, number of patterns learned and accuracy against data points, we also measure the accuracy against patterns, where both the same number of clusters and different number of clusters in ground truth distributions are used.

In chapter 4 (refer to 4.6), we have shown that the extent of overlapping of clusters affects how accurate the learned number of clusters is and accuracy against data points. In the following sections, we show the results on well-separated clusters first, then on overlapped clusters.

5.6.1 Experiment on well-separated clusters

Figure 5.24 shows the result when K-means is used to cluster data points generated from fully symmetric Gaussians [3]. Centralising raw data points to a central agent C_r_R2 takes longer time than other methods. In terms of amount of information received, $S2_A1_R1$ and $S2_A1_R2$ require much more exchanged information. As before, this is because agent has to do repeated clustering whenever receives new information and share it with neighbours.

When the number of clusters in each pattern in the ground truth distribution used to generated data points are the same, the learned number of clusters are close to the ground truth number of clusters (the third row) and estimated number of patterns are close to the ground truth number of patterns (the fourth row). The accuracy on data points against centralised method and ground truth are reaching 98%. This is as expected, since we have shown that our algorithms perform nearly perfectly when clusters are not overlapped. Note that there is little difference in

[3] Since GMM and SG also perform well when clusters are well-separated, so we omit the result with GMM and SG

accuracy on data points and patterns between proposed algorithms. When the number of clusters in each ground truth pattern are different, the accuracy against patterns is improved and the accuracy reaches as high as 99%. This is because the distance between agents with different numbers of clusters in their model is high, and we have already shown that we get high accuracy for detecting the number of clusters, and so it becomes easy to identify agents from different groups.

The last four rows shows the accuracy against patterns by MD1 and MD2 separately. When there is no difference between estimated number of patterns (same or different number of clusters), the accuracy against patterns are similar (see the first columns). The difference occurs when there is a gap between the estimated number of patterns (especially the last column). Compared to accuracy achieved by same number of clusters, accuracy against pattern when there are different number of clusters is round 18% higher.

When data points from partially symmetric Gaussians are used (figure 5.25), the learned number of clusters are close to the ground truth number of clusters and estimated number of patterns (except the case of three patterns) are close to the ground truth patterns. Again, there is little difference between the proposed algorithms. Even when the estimated number of patterns are wrong (the last column), the accuracy against patterns are still above 70%. That is because two sub-patterns are merged incorrectly into a big sub-pattern, and so there is only a slightly decrease of accuracy.

Figure 5.26 shows the result when data points are generated from a uniform distribution. As before, the learned number of clusters and estimated number of patterns are nearly perfect when compared to the ground truth. Again, accuracy against patterns is higher when the number of clusters in the patterns is different (improved by as much as 20%).

To summarise, our proposed method performs well when clusters are well separated. The learned number of clusters and estimated number of patterns are close to the ground truth. In addition, the accuracy on data point and patterns are high, reaching accuracy level of 98%. The performance improves when the number of clusters in the patterns is different.

5.6.2 Experiment on overlapped clusters

In this section, we show the experimental results when there is substantial overlapping between clusters. Again, we only describe the main results and a table of



Figure 5.24: K-means on multiple patterns dense network with fully symmetric well separated clusters



Figure 5.25: K-means on multiple patterns dense networks with partially symmetric well separated clusters

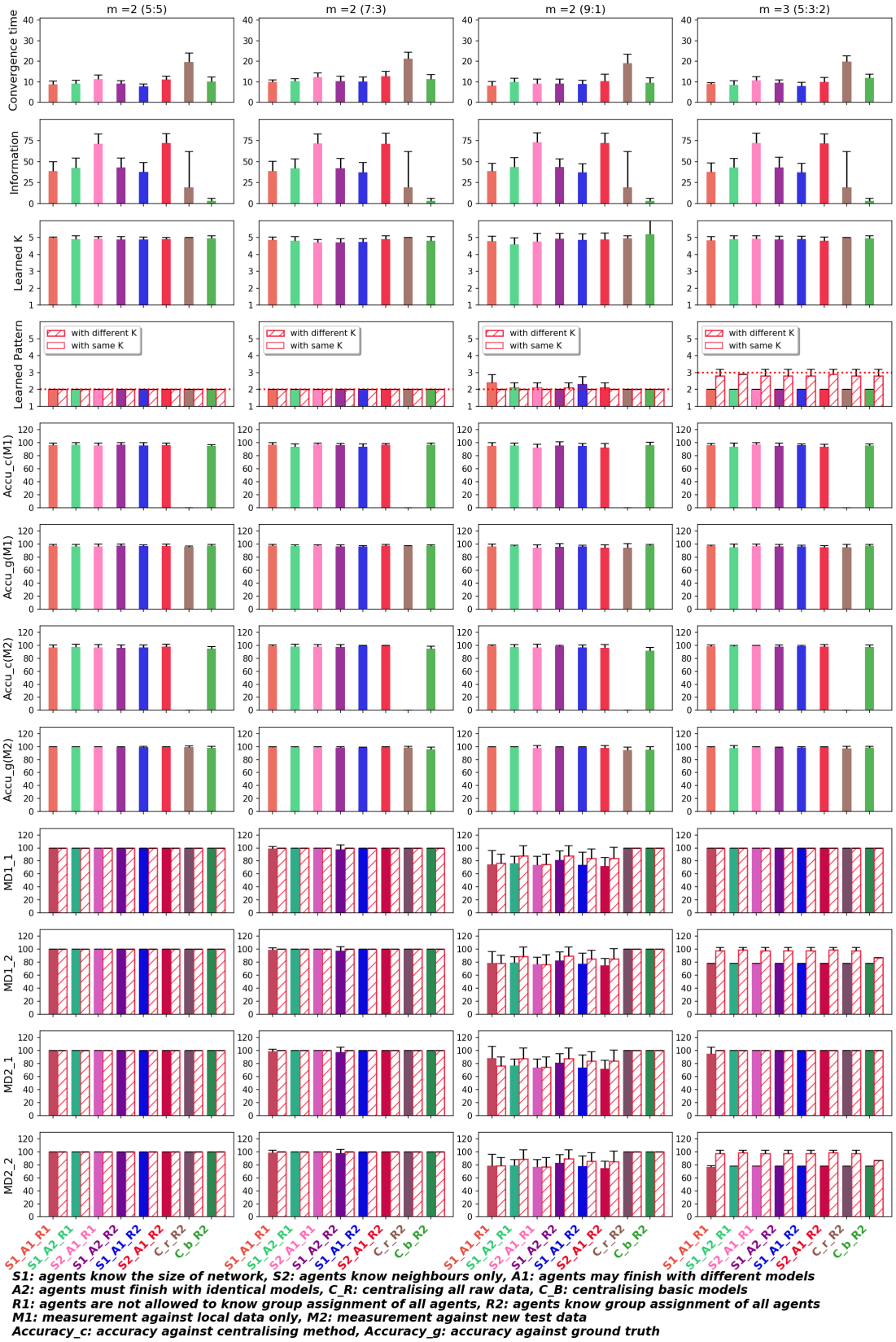


Figure 5.26: K-means on multiple patterns dense networks with uniformly well separated clusters

significant testes will be given at the end [4]. Compared to that with well-separated clusters, we expect that the performance will degrade, based on the results produced in Chapter 4. We only show the results on dense networks. The results on sparse network are similar and are attached in appendix 8.5. In addition, the comparison of two methods V1 and V2 (refer to section 5.3) to predict the similar sub-pattern is also in that appendix, where we show that predicting which sub-pattern an agent is in by comparing the weighted EMD between centroids performs better than the other method.

5.6.2.1 K-means as clustering algorithm

Figure 5.27 shows the results on dense networks when K-means is used as clustering algorithm. Compared to the ground truth number of patterns, there is no significant difference in learned number of patterns between the same and different settings of number of clusters $\boldsymbol{k}$. However, we observe a significant difference in accuracy against patterns(except the case with a singleton agent).

As before, centralising raw data points to a central agent C_r_R2 converges slower than other methods and $S2_A1_R1$ and $S2_A1_R2$ require much more message transmissions. C_r_R2 outperform other methods in predicting the number of clusters. Again, that is because no errors (for instance, caused by regeneration) are introduced compared to the distributed methods.

On the fourth row of figure 5.27, we show the result with same (solid bar) and different (slashed white bar) number of clusters in the patterns. It shows that when there are only two patterns, no matter what the actual split of agents are, both methods can predict the same number of patterns as ground truth. However, the estimated number of patterns are different when there are three patterns (5:3:2). When the number of clusters in the patterns in the ground truth distributions are the same, the algorithms predict there are still two patterns. But the algorithms estimate that the number of patterns are close to 3 (2.8 on average) when the number of clusters in the patterns are different. As before, it is a challenge for the algorithm to separate agents when the number of clusters over patterns are same, and those clusters overlap, so two patterns are merged into a big pattern.

When there is no difference between estimated number of patterns (same or different number of clusters), the accuracy against patterns are similar (see the first

[4] We only report the significant study on dense graph with K-means as clustering algorithm, since the results with GMM and SG are similar.

three columns). The accuracy against pattern is up to 15% higher when the number of patterns is close to the ground truth (see last column). As before, the number of clusters is different, it is easier to detect the different patterns, and so assigning agents to groups is more consistent.

When the underlying datasets are sampled from partially symmetric Gaussians (see figure 5.28), we observe there is no significant difference in learned number of patterns with the setting of different $\boldsymbol{k}$. The learned number of patterns with different $\boldsymbol{k}$ is much likely to close to the ground truth value. However, there is a significant difference in accuracy against patterns between same $\boldsymbol{k}$ and different $\boldsymbol{k}$.

The learned number of pattern with different number of clusters outperforms that by same number of clusters. In the first three columns (5:5, 7:3 and 9:1), the number of patterns are overestimated by the number of clusters is the same for each pattern. As expected, the corresponding accuracy against patterns over the setting of same number of clusters are lower. The reason lies in the fact that K-means may fail in fitting the partially symmetric Gaussians.

Figure 5.29 shows the result with uniformly distributed data. Compared to the ground truth number of patterns, the learned patterns with the same and different $\boldsymbol{k}$ differ slightly. The learned number of patterns by both settings are close to the ground truth value. However, we observed there are substantial difference in accuracy against pattern when the split of agents are 7:3 and 5:3:2.

Again, the number of patterns estimated for the cases with the same number of clusters is overestimated except for the case of three patterns (5:3:2). Note that when there are three sub-patterns (fourth row of last column), it is performance is better on predicting the number of patterns when the k-value is the same. This is different to previous results.

Compared to that with Gaussians, no matter fully or partially symmetric, the accuracy against pattern dropped to around 50% when one of the patterns is a singleton agent (9:1). This caused by the spectral clustering method we used to divide agents into groups, which prefers to generate large clusters. Unless the singleton agent pattern is very distinct, the method will prefer to group that agent with others.



Figure 5.27: K-means on dense network with multiple patterns and symmetric multi-dimensional Gaussians



Figure 5.28: K-means on dense network with multiple patterns and asymmetric multi-dimensional Gaussians

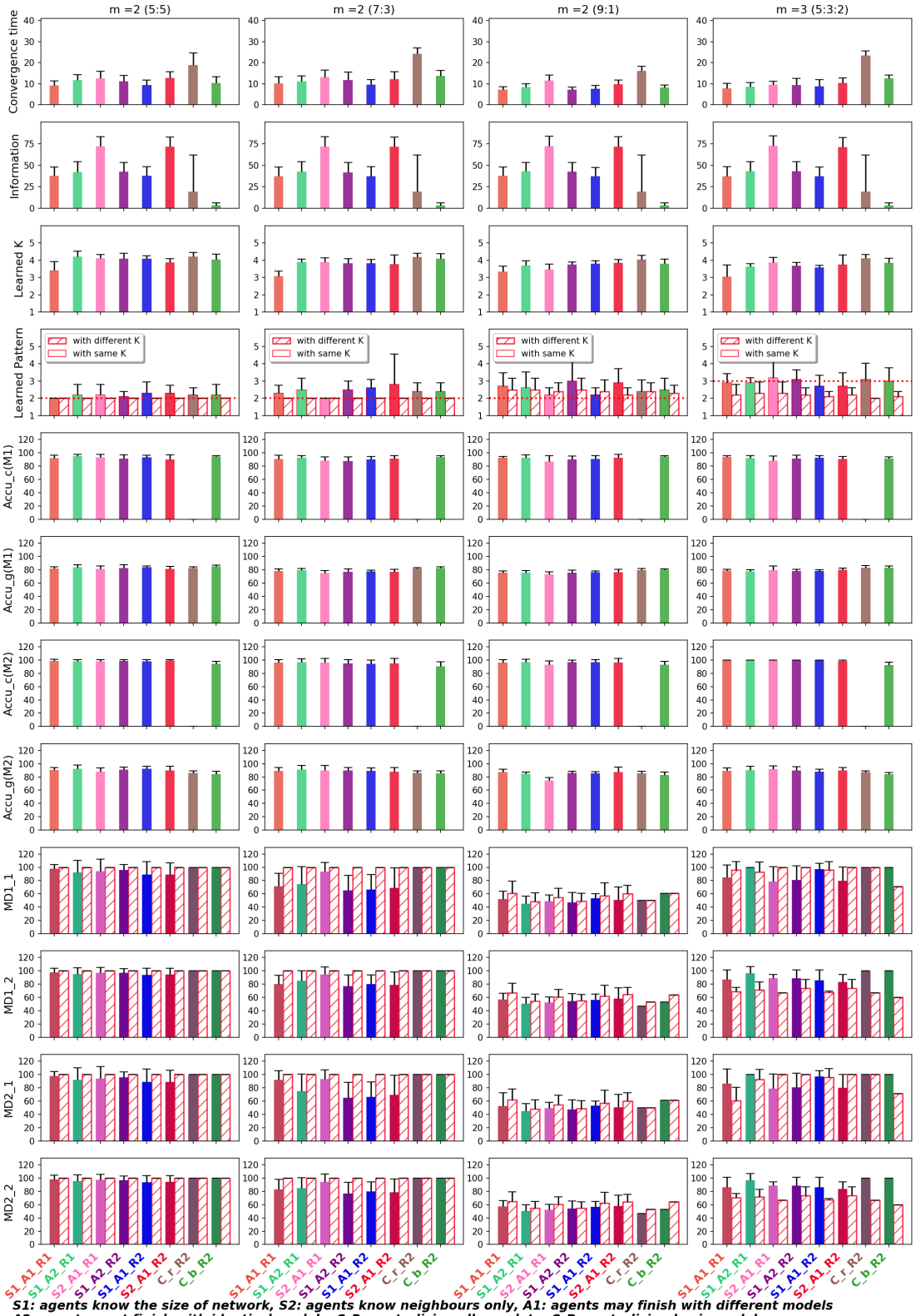


Figure 5.29: K-means on dense network with multiple patterns and uniformly distributed dataset

5.6.2.2 GMM as clustering algorithm

In this section, we show the experimental results when GMM is used as the clustering algorithm. As before, there is no significant difference in learned number of patterns between the same and different settings of number of clusters $\boldsymbol{k}$. However, there is a significant difference in accuracy against patterns(except the case with a singleton agent).

Compared to the results with K-means, the convergence time is much shorter (see the first row of figure 5.30 and figure 5.27), as in previous results. As before, centralising all raw data is slower than other methods and more messages are required if agents are only aware of their neighbours.

Figure 5.30 shows the results when the underlying datasets are generated from fully symmetric Gaussians. The results show the expected behaviour.

Figure 5.31 shows the result with data points generated from partially symmetric Gaussians as underlying datasets. Again, the results are similar as that using k-means as the clustering algorithm.

When uniformly distributed data is used, the number of patterns learned by different number of clusters is much more accurate (refer to figure 5.32) than when using K-means. Thus, the corresponding accuracy against pattern is higher. When there is a singleton agent (9:1), the accuracy against pattern dropped noticeably to the other types of ground truth data.

5.6.2.3 SG as clustering algorithm

In this section, we show the experimental result when SG is used as the clustering algorithm. Similarly, there is no significant difference in learned number of patterns between the same and different settings of number of clusters $\boldsymbol{k}$. However, we observe a significant difference in accuracy against patterns(except the case with a singleton agent).

Figure 5.33, figure 5.34 and figure 5.35 show the result on data points generated from fully symmetric Gaussians, partially symmetric Gaussians and uniform distribution, separately. The result is similar to that with GMM as clustering algorithm.



Figure 5.30: GMM on dense network with multiple patterns and symmetric multi-dimensional Gaussians



Figure 5.31: GMM on dense network with multiple patterns and asymmetric multi-dimensional Gaussians

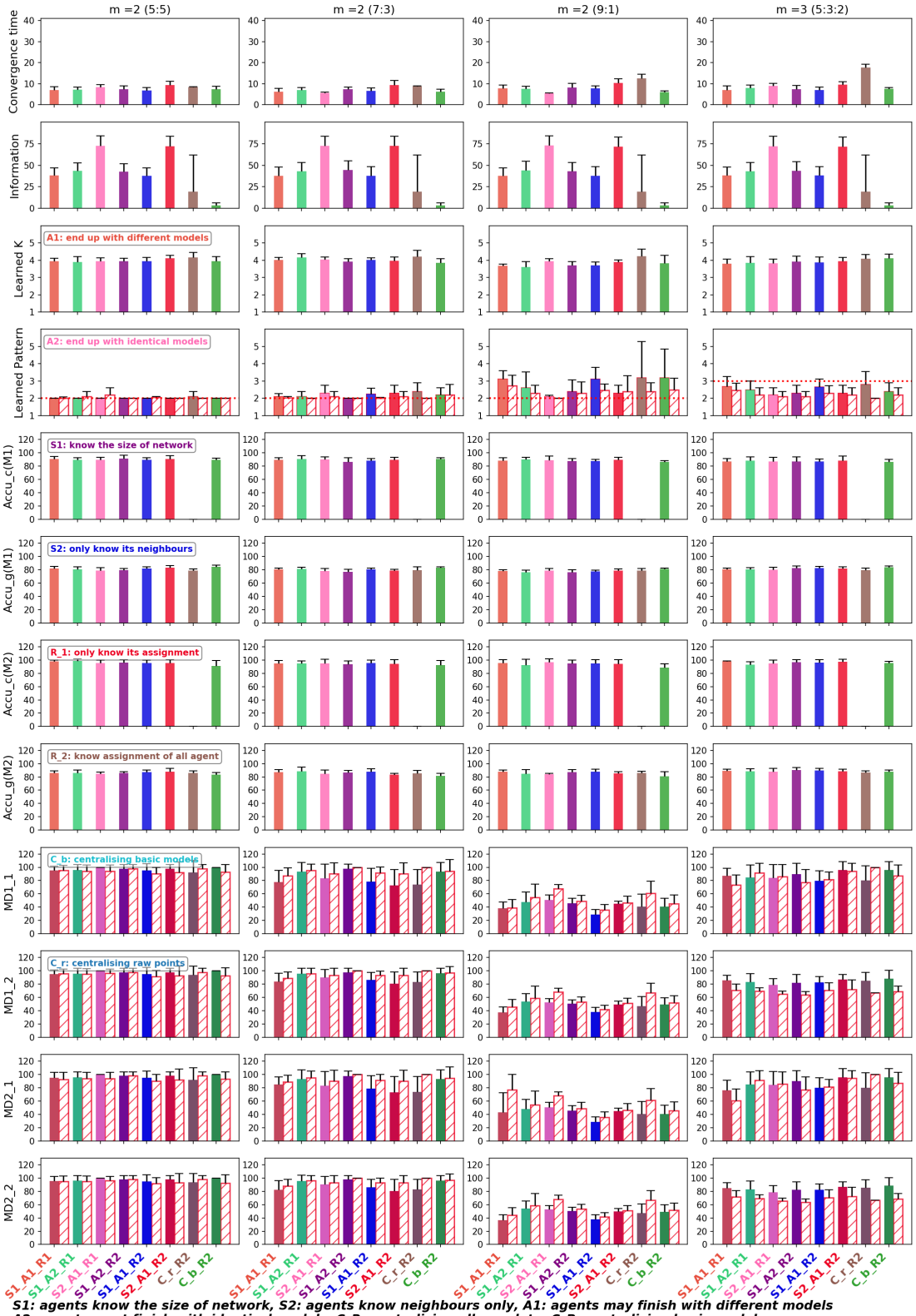


Figure 5.32: GMM on dense network with multiple patterns and uniformly distributed dataset



Figure 5.33: SG on dense network with multiple patterns and fully symmetric multi-dimensional Gaussians

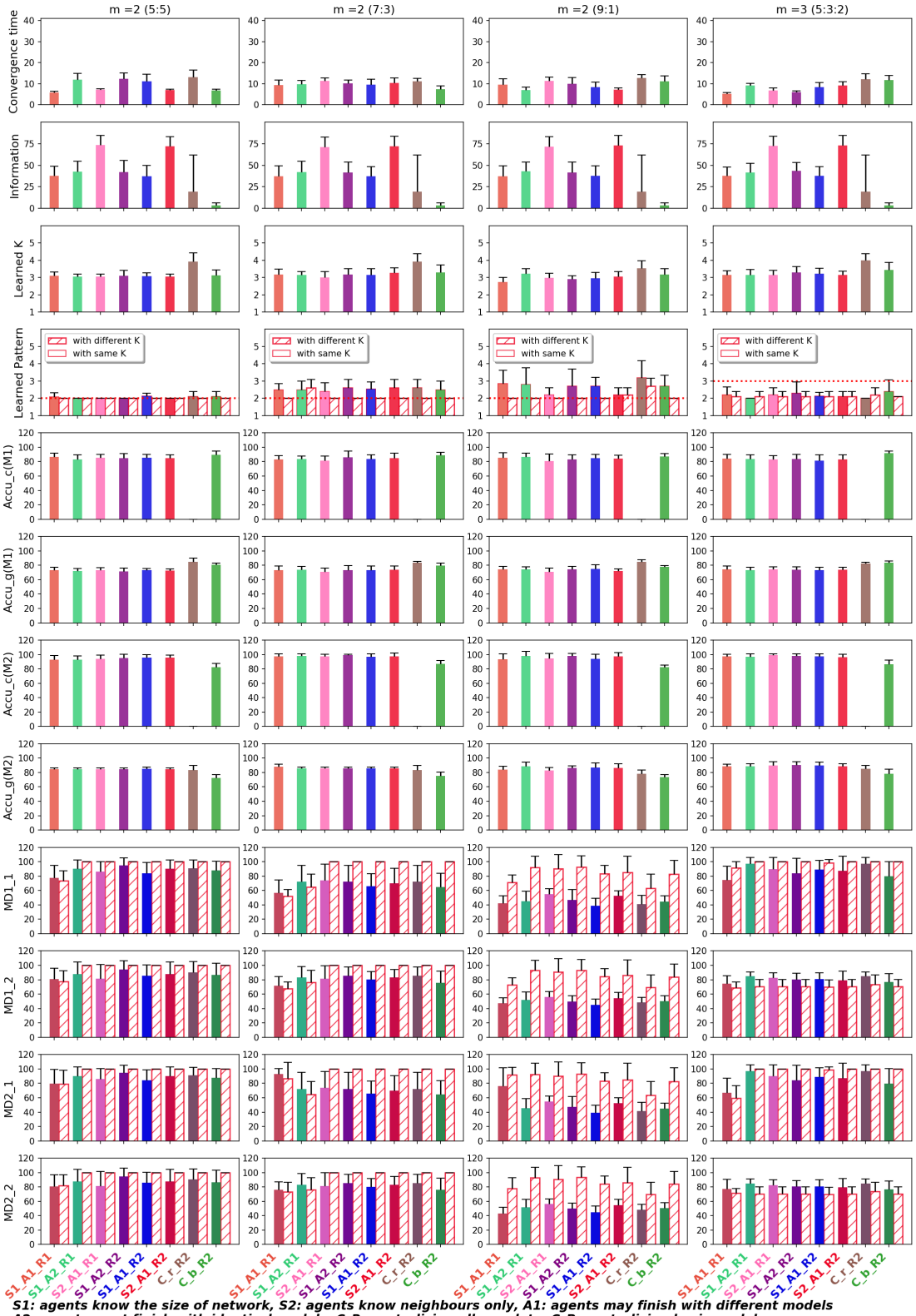


Figure 5.34: SG on dense network with multiple patterns and partially symmetric multi-dimensional Gaussians

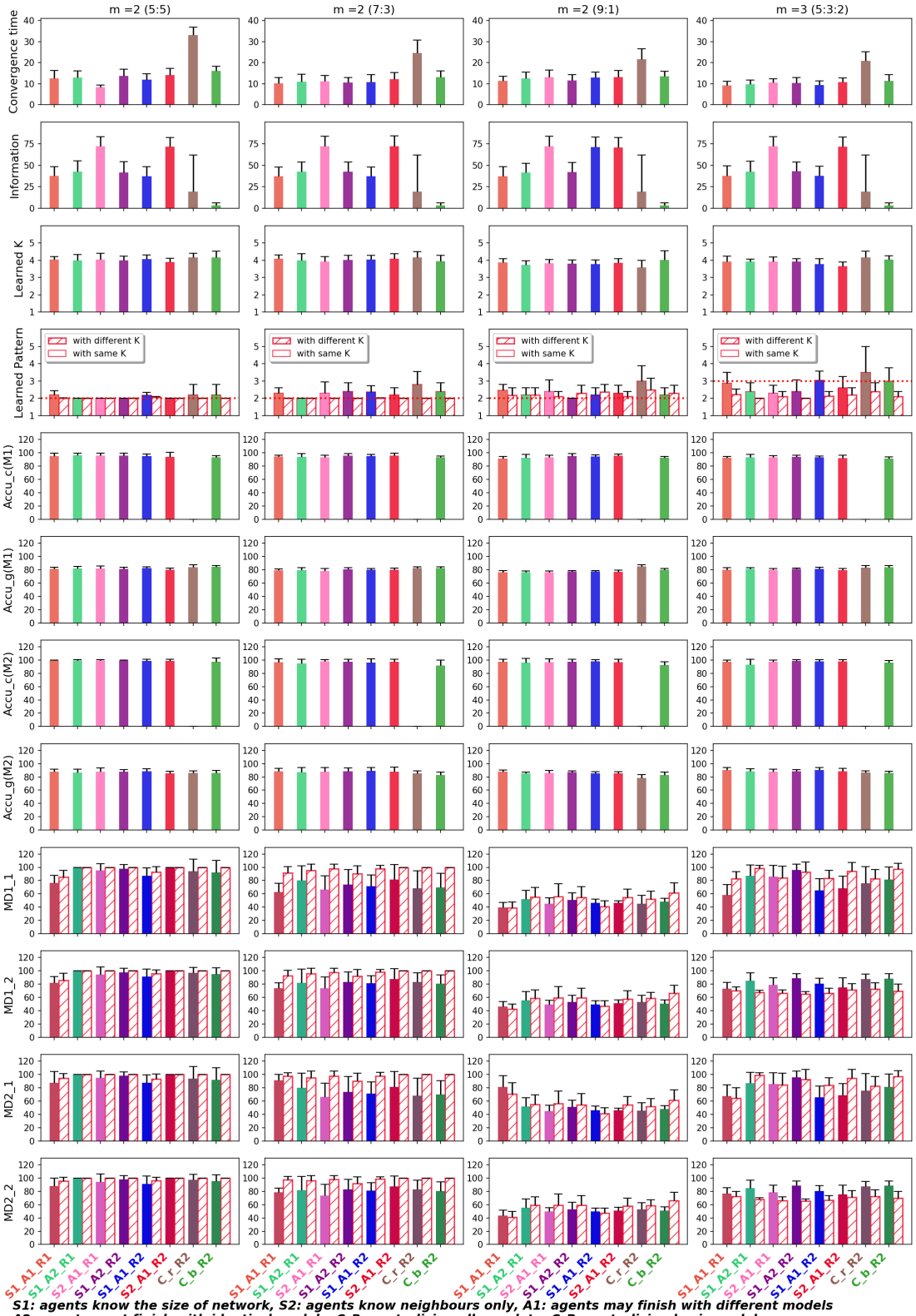


Figure 5.35: SG on dense network with multiple patterns and uniformly distributed dataset

5.6.3 Summary

In this section, first we compared the result on overlapped clusters with well-separated clusters. When the clusters overlap with each other, the learned number of clusters is smaller and further away from ground truth, than that with well-separated clusters. This was expected because data points in the boundary of two clusters mean that clusters will tend to be merged together. In addition, the learned number of patterns is less accurate and the corresponding accuracy against pattern is as much as 20% lower. Note that the accuracy against pattern is above 80% and can reach 99% even there is a singleton agent when clusters are well separated. It indicates that the underlying datasets affect the performance of proposed algorithms.

When the patterns have the same number of clusters, the performance degrades when there are overlapping between clusters. The accuracy against pattern dropped noticeably, especially when there is a singleton agent (9:1). When the number of clusters is different, it is easier to detect the different patterns, and so assigning agents to groups is more consistent.

Figure 5.29, 5.32 and 5.35 show that accuracy against data points is higher when K-means is used to cluster the underlying data points generated from a uniform distribution. That is because bounding boxes outperforms SG and GMM over the regeneration. When the underlying dataset is uniformly distributed, data points are spread out rather than gathered together. Note that the accuracy against pattern is the worst when there is a singleton agent (around 40%). That is because spectral clustering used to divide the agents into separated groups prefers large groups when the similarity between agents is not so distinct.

5.7 Increasing accuracy against pattern when there is a singleton agent

In last section, we showed the experimental result when clusters are well separated and overlapped. When there are some overlapping between clusters, the accuracy against pattern is low if only one agent is sampling data from different distributions (9:1). That is because the similarity between agents are quite high and it makes the algorithm difficult to tell the singleton agent from the rest of agents. In spectral clustering, the similarity and size of groups play an important role in grouping and it prefers large clusters. This raises the question of how to

boost the accuracy against pattern when the similarity between agents is high and there are very small agent groups?

5.7.1 Cluster model between different agents are similar

First, we consider the scenario that the cluster models are similar. The main idea is to detect the singleton agent before proceeding to the spectral clustering. The singleton agent is detected by measuring the deviation of each agent from the rest of agents. After that, the deviation based distance (DBD) is measured by IQR. IQR is defined as the difference between Q3 and Q1. Data points that locate outside [Q1- 1.5*IQR, Q3+1.5*IQR] are considered as outliers.

Figure 5.36 shows the performance achieved by DBD when K-means is used to fit data points generated from partially symmetric Gaussians. The left column m shows the performance without DBD method, the middle column $\overline{m}$ shows the performance with DBD method, and the right column shows the difference between them (the difference of solid bar and slash bar between column m and column $\overline{m}$ are measured by D1 and D2 respectively). There is little difference in convergence time, information, learned number of patterns and learned number of clusters. The accuracy against patterns is increased by around 10% on average by the DBD method. Note that the DBD method has little effect on the accuracy against patterns when the number of clusters over multiple patterns is different (the accuracy decreased by around 2%). This appears to be because the number of patterns estimated on the middle column is worse, which cause the corresponding accuracy against pattern to drop as well.

Figure 5.37 shows that performance by DBD method when K-means is used to fit the uniformly distributed dataset. Again, there is little difference in measurement except for accuracy against patterns. When the number of clusters is the same, the accuracy against patterns is improved by as much as 8% on average. The accuracy against patterns is also slightly increased when the number of clusters over patterns is different (less than 1% on average). The most likely reason for this is that the similarity between agents is low in this case and detecting the singleton agent is not needed.

Figure 5.38 and 5.39 show the performance when GMM is used as clustering algorithm. The results are similar to those using k-means as the clustering algorithm.

Figure 5.40 and 5.41 show the comparison result when SG is used as the clustering algorithm. Again, the results show the same behaviour.

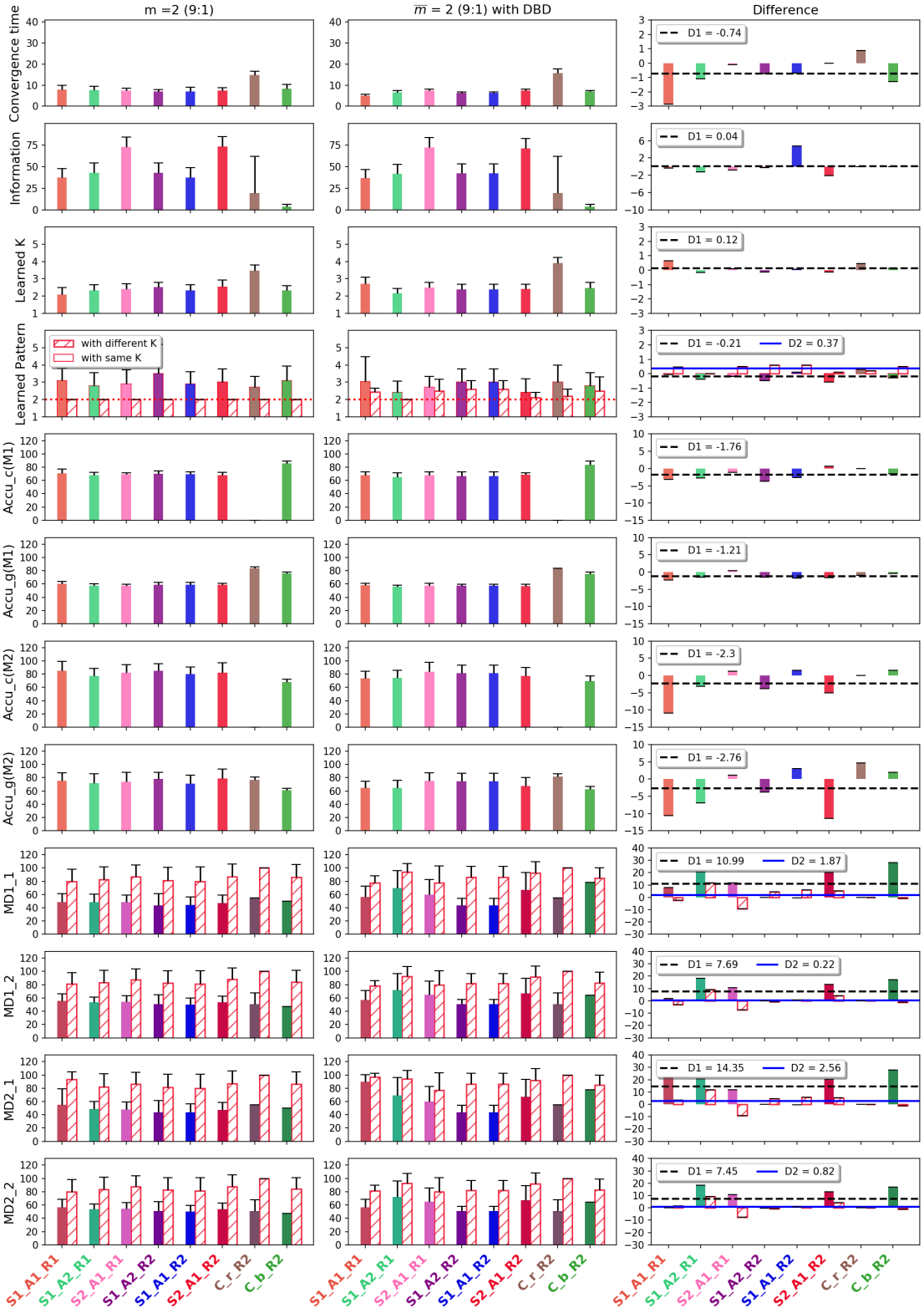


Figure 5.36: Performance achieved by DBD with K-means as clustering algorithm and asymmetric Gaussians as underlying dataset

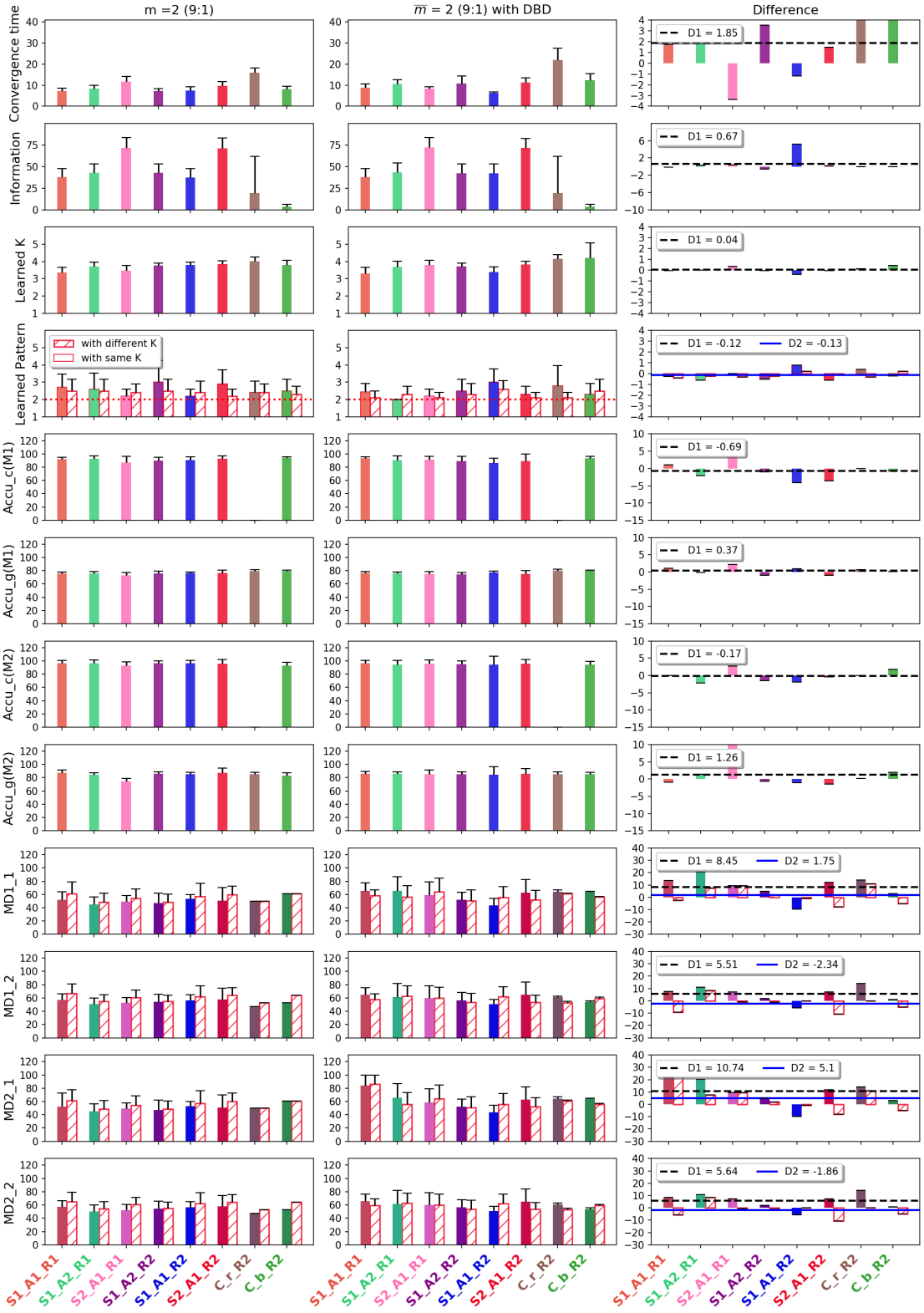


Figure 5.37: Performance achieved by DBD with K-means as clustering algorithm and uniformly distributed dataset as as underlying dataset

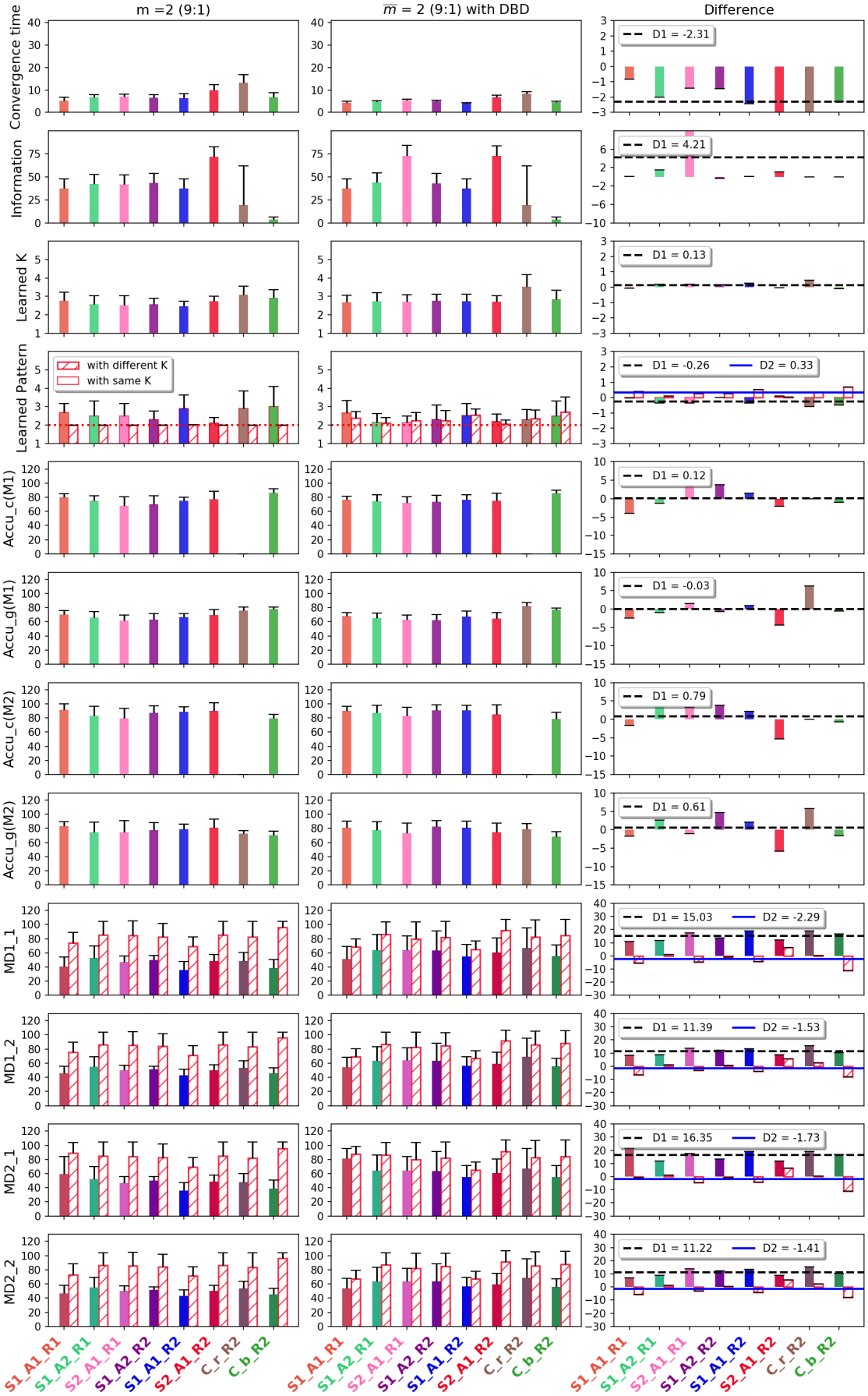


Figure 5.38: Performance achieved by DBD with GMM as clustering algorithm and asymmetric Gaussians as underlying dataset

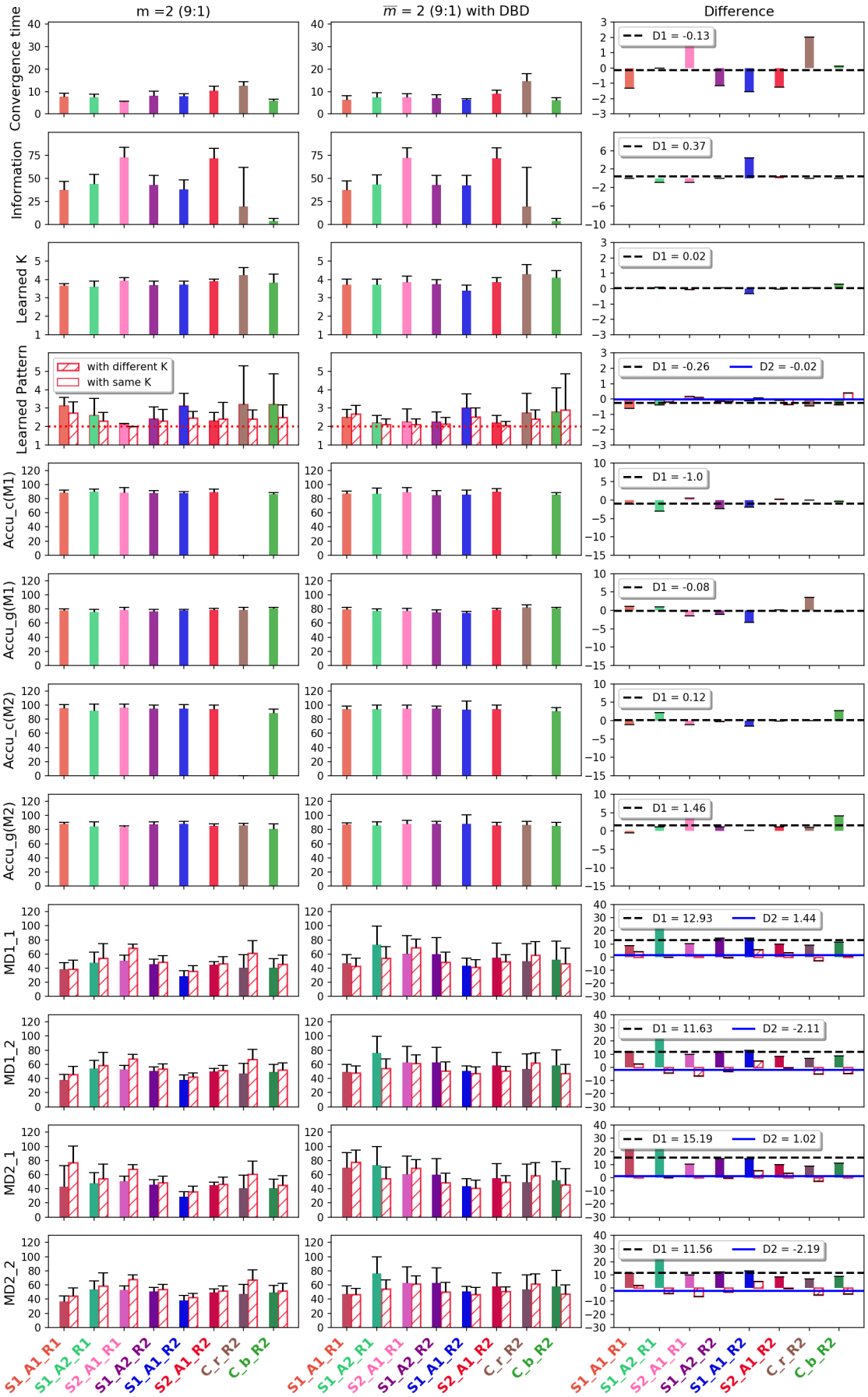


Figure 5.39: Performance achieved by DBD with GMM as clustering algorithm and uniformly distributed datasets as underlying dataset

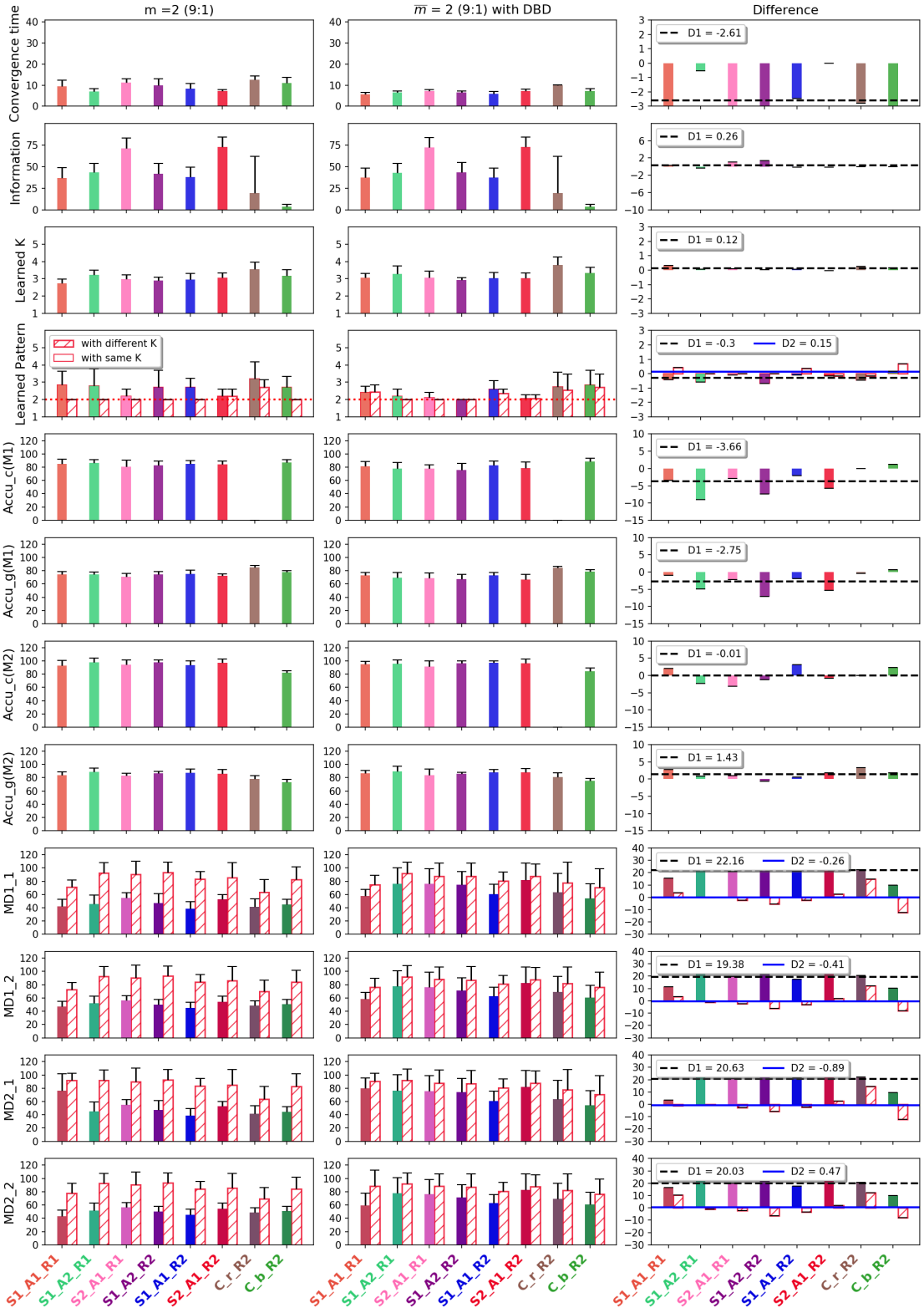


Figure 5.40: Performance achieved by DBD with SG as clustering algorithm and asymmetric Gaussians as underlying dataset



Figure 5.41: Performance achieved by DBD with SG as clustering algorithm and uniformly distributed datasets as underlying dataset

To summarise, the low accuracy against patterns caused by a singleton agent pattern could be improved when: 1) the proposed DBD method is applied when the number of cluster over ground truth distribution are same, or 2) the number of clusters over ground truth distribution is different.

5.7.2 Cluster model between different agents are dissimilar

In last section, we considered the case that cluster models between different agents are similar. In this section, we consider the case that cluster models are dissimilar. Note that cluster models with different number of clusters over ground truth distributions has been considered in preceding sections. We now consider cluster models with the same number of clusters but the centroids of clusters differ substantially in locations compared to other patterns in this section.

We focus on the case of three patterns (5:3:2) and underlying data points are generated from fully symmetric Gaussians. Previously, the ground truth distribution for the second sub-pattern and third sub-pattern is similar. We rotate the ground truth distribution for the second-pattern around the grid, and create a different ground truth distribution. The agents in the third sub-pattern sample data points from this rotated and different ground truth distribution. Figure 5.42 shows one example of dataset before and after rotation, with a noticeable difference between the two patterns. It shows that the dataset differs noticeably from the previous dataset after rotation. However, the number of clusters still remains the same. Again, the experiment is tested on 10-agent network and different assumptions and requirement are applied.

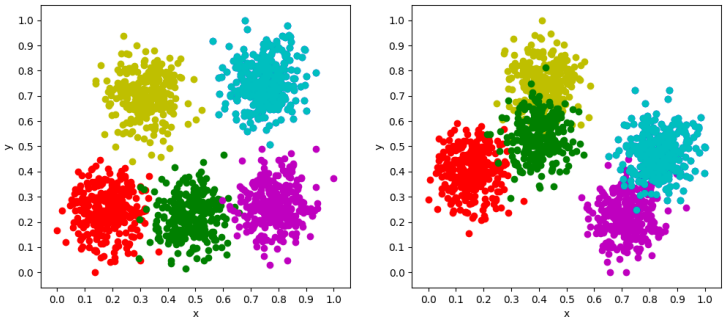


Figure 5.42: Dataset before and after rotation (left to right)

Figure 5.43 shows the accuracy against patterns with different levels of similarities of the data distribution in the patterns. There is little difference in accuracy

against pattern M_{e1} between similar and dissimilar agents (blue and green line). It indicates that agents are put into the same group as they were in for the ground truth. When accuracy against pattern M_{e2} is computed, accuracy is much higher when ground truth data distributions are dissimilar. That is because two similar patterns are merged into a big pattern if agents are similar.

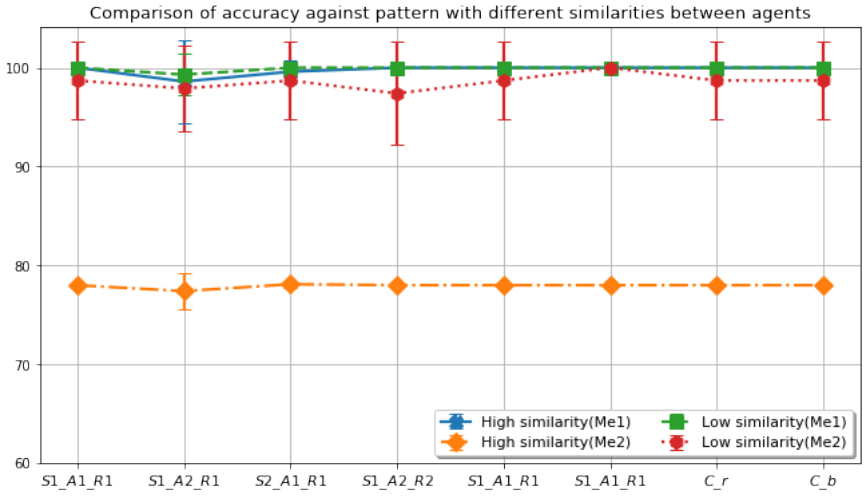


Figure 5.43: Accuracy against patterns with different similarities between agents

Figure 5.44 shows the corresponding number of clusters and patterns learned. The number of clusters learned is much closer to ground truth number of clusters when agents are dissimilar. In addition, the learned number of patterns is much closer to the ground truth. Based on the result of accuracy, number of clusters and patterns, we can infer that two small clusters are merged to a big cluster when agents are similar. It is challenge for the algorithm to distinguish these two clusters. However, those two small clusters can be separated when agents are dissimilar.

To summarise, the underlying datasets, for instance, the overlapping between clusters and the ground truth distributions that used to generate data points, play a vital role in the performance of proposed algorithms. When the clusters are well separated, our proposed algorithms perform well. The learned number of patterns and number of clusters are close to the ground truth. The performance degrades when there is some overlapping between clusters and one question occurs when there is a singleton agent: the accuracy against patterns are low. This question can be partially addressed by the proposed DBD method, where the accuracy against patterns are increased by as much as 20%.

Moreover, our proposed methods are capable of separating agents that differ substantially from each other even when the number of clusters are same. In

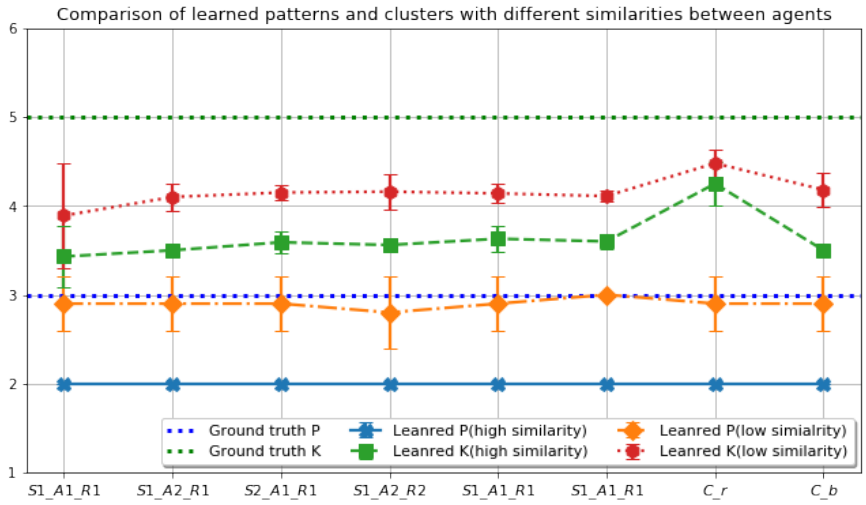


Figure 5.44: Accuracy against patterns with different similarities between agents

addition, our proposed methods perform well when agents have different number of clusters. Our method only struggles when agents are highly similar to each other (both the number of clusters and location of centroids are same).

5.8 Significance Study

In this section, we show the significance study of experimental results. Again, we use the Shapiro-Wilk's test to test the assumption of normality. Results show that not all assumptions are hold, but we still show the significant study for all scenarios.

Table 5.3 shows the one sample t-test for learned number of patterns with the same number of clusters and different number of clusters. The Nan value in the table means all the values in the testing list are identical and equal to the test mean while the inf value means all the values in the testing list are identical and not equal to the test mean. It shows that the estimated mean is much likely to be equal to the test mean if the number of clusters $\boldsymbol{k}$ over patterns is different (the corresponding p-value is Nan or greater than 0.05).

Table 5.4 shows the tukey test for accuracy against patterns with the same $\boldsymbol{k}$ and different $\boldsymbol{k}$. It shows that there is significant difference in accuracy against patterns with the same $\boldsymbol{k}$ and different $\boldsymbol{k}$ (most p-values is less than 0.05), except the case that there is a singleton agent (9:1). As we mentioned before, the most possible reason is that the spectral clustering, we used to detect the patterns, prefers large size of patterns. It is a challenge to detect the patterns with a

Table 5.3: One sample t-test for learned number of patterns, only the results on dense graph with K-means as clustering algorithm is compared

	Symmetric Gaussians				Asymmetric Gaussians			
Learned Patterns with same k	5:5 ($\diamond$, P value)	7:3 ($\diamond$, P value)	9:1 ($\diamond$, P value)	5:3:2 ($\diamond$, P value)	5:5 ($\diamond$, P value)	7:3 ($\diamond$, P value)	9:1 ($\diamond$, P value)	5:3:2 ($\diamond$, P value)
$S1_A1_R1$	(Nan, Nan)	(Nan, Nan)	(Nan, Nan)	(-inf, 0.0)	(1.0, 0.343)	(3.674, 0.005)	(4.714, 0.001)	(-3.28, 0.01)
$S1_A2_R1$	(Nan, Nan)	(Nan, Nan)	(Nan, Nan)	(-inf, 0.0)	(1.0, 0.343)	(4.583, 0.001)	(3.207, 0.011)	(-4.583, 0.001)
$S2_A1_R1$	(Nan, Nan)	(Nan, Nan)	(Nan, Nan)	(-inf, 0.0)	(1.5, 0.168)	(4.583, 0.001)	(3.25, 0.01)	(-2.714, 0.024)
$S1_A2_R2$	(Nan, Nan)	(Nan, Nan)	(Nan, Nan)	(-inf, 0.0)	(1.5, 0.168)	(4.583, 0.001)	(1.406, 0.193)	(-3.674, 0.005)
$S1_A1_R2$	(Nan, Nan)	(Nan, Nan)	(Nan, Nan)	(-inf, 0.0)	(1.964, 0.081)	(3.674, 0.005)	(2.666, 0.026)	(-3.28, 0.01)
$S2_A1_R2$	(Nan, Nan)	(Nan, Nan)	(Nan, Nan)	(-inf, 0.0)	(1.964, 0.081)	(3.674, 0.005)	(3.873, 0.004)	(-6.0, 0.0)
C_r_R2	(Nan, Nan)	(Nan, Nan)	(Nan, Nan)	(-inf, 0.0)	(2.449, 0.037)	(4.0, 0.003)	(3.28, 0.01)	(-3.28, 0.01)
C_b_R2	(Nan, Nan)	(Nan, Nan)	(Nan, Nan)	(-inf, 0.0)	(1.0, 0.343)	(2.449, 0.037)	(4.265, 0.0)	(-inf, 0.0)

	Uniform distributions			
Learned Patterns with same k	5:5 ($\diamond$, P value)	7:3 ($\diamond$, P value)	9:1 ($\diamond$, P value)	5:3:2 ($\diamond$, P value)
$S1_A1_R1$	(Nan, Nan)	(1.964, 0.081)	(2.689, 0.025)	(-0.557, 0.591)
$S1_A2_R1$	(1.0, 0.343)	(2.236, 0.052)	(1.964, 0.081)	(-1.0, 0.343)
$S2_A1_R1$	(1.0, 0.343)	(Nan, Nan)	(1.964, 0.081)	(0.612, 0.555)
$S1_A2_R2$	(Nan, Nan)	(3.0, 0.015)	(2.372, 0.042)	(0.557, 0.591)
$S1_A1_R2$	(1.0, 0.343)	(3.674, 0.005)	(1.5, 0.168)	(-1.406, 0.193)
$S2_A1_R2$	(1.5, 0.168)	(1.35, 0.21)	(3.25, 0.01)	(-1.152, 0.279)
C_b_R2	(1.5, 0.168)	(2.449, 0.037)	(1.809, 0.104)	(0.318, 0.758)
C_r_R2	(1.0, 0.343)	(2.449, 0.037)	(2.236, 0.052)	(0.0, 1.0)

	Symmetric Gaussians				Asymmetric Gaussians			
Learned Patterns with different k	5:5 ($\diamond$, P value)	7:3 ($\diamond$, P value)	9:1 ($\diamond$, P value)	5:3:2 ($\diamond$, P value)	5:5 ($\diamond$, P value)	7:3 ($\diamond$, P value)	9:1 ($\diamond$, P value)	5:3:2 ($\diamond$, P value)
$S1_A1_R1$	(Nan, Nan)	(Nan, Nan)	(Nan, Nan)	(-1.5, 0.168)	(Nan, Nan)	(Nan, Nan)	(Nan, Nan)	(-9.0, 0.0)
$S1_A2_R1$	(Nan, Nan)	(Nan, Nan)	(Nan, Nan)	(-1.5, 0.168)	(Nan, Nan)	(Nan, Nan)	(Nan, Nan)	(-9.0, 0.0)
$S2_A1_R1$	(Nan, Nan)	(Nan, Nan)	(Nan, Nan)	(-1.5, 0.168)	(Nan, Nan)	(Nan, Nan)	(Nan, Nan)	(-12.696, 0.0)
$S1_A2_R1$	(Nan, Nan)	(Nan, Nan)	(Nan, Nan)	(-1.5, 0.168)	(Nan, Nan)	(Nan, Nan)	(Nan, Nan)	(-9.0, 0.0)
$S2_A2_R2$	(Nan, Nan)	(Nan, Nan)	(Nan, Nan)	(-1.5, 0.168)	(Nan, Nan)	(Nan, Nan)	(Nan, Nan)	(-9.0,0.0)
$S1_A1_R2$	(Nan, Nan)	(Nan, Nan)	(Nan, Nan)	(-1.5, 0.168)	(Nan, Nan)	(Nan, Nan)	(Nan, Nan)	(-9.0, 0.0)
$S2_A1_R2$	(Nan, Nan)	(Nan, Nan)	(Nan, Nan)	(-1.5, 0.168)	(Nan, Nan)	(Nan, Nan)	(Nan, Nan)	(-9.0, 0.0)
C_r_R2	(Nan, Nan)	(Nan, Nan)	(Nan, Nan)	(-1.5, 0.168)	(Nan, Nan)	(Nan, Nan)	(Nan, Nan)	(-9.0,0.0)
C_b_R2	(Nan, Nan)	(Nan, Nan)	(Nan, Nan)	(-1.5, 0.168)	(Nan, Nan)	(Nan, Nan)	(Nan, Nan)	(-9.0,0.0)

	Uniform distributions			
Learned Patterns with different k	5:5 ($\diamond$, P value)	7:3 ($\diamond$, P value)	9:1 ($\diamond$, P value)	5:3:2 ($\diamond$, P value)
$S1_A1_R1$	(Nan, Nan)	(Nan, Nan)	(2.236, 0.052)	(-4.0, 0.003)
$S1_A2_R1$	(Nan, Nan)	(Nan, Nan)	(2.236, 0.052)	(-3.28, 0.01)
$S2_A1_R1$	(Nan, Nan)	(Nan, Nan)	(2.449, 0.037)	(-inf, 0.0)
$S1_A2_R1$	(Nan, Nan)	(Nan, Nan)	(2.236, 0.052)	(-6.0, 0.0)
$S1_A1_R2$	(Nan, Nan)	(Nan, Nan)	(1.809, 0.104)	(-6.0, 0.0)
$S2_A1_R2$	(Nan, Nan)	(Nan, Nan)	(1.5, 0.168)	(0.31, 0.83)
$S1_A1_R1$	(Nan, Nan)	(Nan, Nan)	(1.5, 0.168)	(-6.0, 0.0)
C_b_R2	(Nan, Nan)	(Nan, Nan)	(2.449, 0.037)	(-inf, 0.0)
C_r_R2	(Nan, Nan)	(Nan, Nan)	(1.964, 0.081)	(-6.0, 0.0)

singleton agent.

5.9 Conclusion

In this chapter, we relax the assumption that all agents are receiving data from the same source and instead we assume that there are multiple different patterns of data being observed by different subgroups of agents. The aim was for agents to be able to identify the different patterns and their associated clusters, rather than combine the multiple patterns into a single description. To reduce the computational effort in assessing similarity of cluster models, we proposed two EMD based methods Weighted EMD and Reversible EMD to measure the similarity between cluster models and infer the number of patterns based on the

Table 5.4: Tukey test for accuracy against patterns with same $\boldsymbol{k}$ and different $\boldsymbol{k}$. Only the results on dense graph with K-means as clustering algorithm is reported. $\diamond$ denotes the difference of mean

	Symmetric Gaussians				**Asymmetric Gaussians**			
Accuracy	5:5 ($\diamond$, P value)	7:3 ($\diamond$, P value)	9:1 ($\diamond$, P value)	5:3:2 ($\diamond$, P value)	5:5 ($\diamond$, P value)	7:3 ($\diamond$, P value)	9:1 ($\diamond$, P value)	5:3:2 ($\diamond$, P value)
*S*1_*A*1_*R*1	(Nan, Nan)	(Nan, Nan)	(1.053, 0.306)	(-1.275, 0.218)	(-3.076, 0.007)	(-5.293, 0.0)	(-6.764, 0.0)	(2.913, 0.042)
*S*1_*A*2_*R*1	(Nan, Nan)	(Nan, Nan)	(-1.129, 0.274)	(-2.512, 0.022)	(-3.211, 0.005)	(-3.747,0.001)	(-3.887, 0.001)	(1.071, 0.299)
*S*2_*A*1_*R*1	(Nan, Nan)	(Nan, Nan)	(-0.236, 0.816)	(-5.507, 0.0)	(-4.004, 0.001)	(-3.144, 0.006)	(-2.832, 0.011)	(1.864, 0.079)
*S*1_*A*2_*R*2	(Nan, Nan)	(Nan, Nan)	(-0.062, 0.951)	(-2.512, 0.022)	(-2.722, 0.036)	(-4.554, 0.0)	(-4.223, 0.001)	(8.577, 0.0)
*S*1_*A*1_*R*2	(Nan, Nan)	(Nan, Nan)	(-0.313, 0.758)	(-2.512, 0.022)	(-2.722, 0.014)	(-4.619, 0.0)	(-4.817, 0.0)	(3.066, 0.007)
*S*2_*A*1_*R*2	(Nan, Nan)	(Nan, Nan)	(-1.964, 0.065)	(-1.206, 0.243)	(-2.99, 0.008)	(-3.669, 0.002)	(-4.271, 0.0)	(1.862, 0.079)
*C*_*r*_*R*2	(Nan, Nan)	(Nan, Nan)	(0.493, 0.628)	(0.156, 0.878)	(-3.344, 0.004)	(-4.359, 0.0)	(-8.833, 0.0)	(2.629, 0.017)
*C*_*b*_*R*2	(Nan, Nan)	(Nan, Nan)	(0.087, 0.932)	(nan,nan)	(-2.194, 0.041)	(-3.047, 0.007)	(-8.258, 0.0)	(2.973, 0.008)

	Uniform distributions			
Accuracy	5:5 ($\diamond$, P value)	7:3 ($\diamond$, P value)	9:1 ($\diamond$, P value)	5:3:2 ($\diamond$, P value)
*S*1_*A*1_*R*1	(-2.013, 0.059)	(-4.579, 0.0)	(-1.712, 0.104)	(3.505, 0.003)
*S*1_*A*2_*R*1	(0.071, 0.945)	(-2.874, 0.01)	(-1.683, 0.11)	(4.773, 0.0)
*S*2_*A*1_*R*1	(-1.923, 0.07)	(-1.837, 0.083)	(-0.318, 0.75)	(2.173, 0.0)
*S*1_*A*2_*R*2	(-1.381, 0.184)	(-4.092, 0.001)	(-0.103, 0.919)	(2.539, 0.021)
*S*1_*A*1_*R*2	(-1.859, 0.08)	(-4.252, 0.0)	(-0.938, 0.36)	(3.485, 0.003)
*S*2_*A*1_*R*2	(-2.187, 0.042)	(-3.752, 0.001)	(0.098, 0.923)	(2.19, 0.042)
*C*_*b*_*R*2	(-1.934, 0.069)	(-2.417, 0.027)	(-1.642,0.118)	(12.321, 0.0)
*C*_*r*_*R*2	(-1.85, 0.081)	(-3.442, 0.003)	(-2.719, 0.014)	(3.6, 0.002)

computed similarity. The experiments shows that weighted EMD is a potential way to measure the similarity between agents. The algorithms that use it perform well in terms of accuracy against centralised method, ground truth and pattern, but they converge faster than when using standard EMD. Compared to the state of art distance measures, it converges faster than the robust EMD and slightly slower than Wavelet EMD, but it achieves higher accuracy than robust EMD and wavelet EMD.

In addition, the similarity between agent cluster models plays a vital role in the prediction of sub-patterns. The proposed algorithm performs well when agent models are dissimilar, or the number of clusters are the same and the number of clusters for agents over patterns is different. The only scenario that the proposed algorithm struggles is when agents' cluster models are similar to each other in both the number of clusters and the locations of centroids. In that case, it is challenge for the algorithm to predict the number of sub-patterns as ground truth patterns.

Moreover, the underlying datasets (for instance, the extent of overlapping between clusters and the ground truth distributions used to generate data points) affect the performance of proposed algorithms. When the clusters are well separated, our proposed algorithms perform well. The learned number of patterns and number of clusters are close to the ground truth, and accuracy on data points and patterns are nearly 98%. The performance degrades when there is some overlapping between clusters and one question occurs when there is a singleton agent:

the accuracy against patterns are low. This question could be partially addressed by the proposed DBD method, where the accuracy against patterns is increased by as much as 20%.

Overall, the cluster models learned by individual agents are accurate, no matter whether they are in single pattern or multiple patterns. But whether the estimated number of patterns are correct depends on the extent of overlapping between clusters and how similar of the clusters models learned by individual agents.

Chapter 6

Detecting single or multiple patterns

In preceding chapters, we focused on in-network clustering for wireless mesh networks. Agents do clustering of their local data and share the summary description with neighbours. Agents must combine the summary descriptions, and eventually all agents should end up with the same understanding of the environment (i.e. of the global data patterns). In some cases, there may be two or more distinct patterns, or distributions of data, each received by distinct groups of agents. We presented methods allowing agents to infer the number of patterns, to group agents according to the patterns they received, and to generate appropriate cluster models for each pattern. Note that the proposed methods do not require the number of clusters as input, nor did they need to know the number of pattens in advance. However, the methods did assume that there was more than one pattern. That is not always justified, and so there is a need for methods which are able to distinguish between an environment with a single pattern (which may still contain multiple clusters) and one with multiple patterns. In this chapter, we tackle that problem and develop such a method, and evaluate it in the same style as previous chapters.

The outline of this chapter is as follows: in section 6.1 we show the research problem and the main algorithm to address that problem. Then two types of methods to detect the number of patterns are proposed in section 6.2 and 6.3. After that, the comparison results are shown in section 6.4, and the conclusion is given in section 6.6.

6.1 Research Problem

Flow chart 6.1 shows the algorithm when we relax the assumptions that the number of patterns is known at the start. Whenever individual agent i receives all information (scenario S1 where the size of the network is known) or new information (scenario S2 where each agent is only aware of its neighbours) from its neighbours, it uses that information to decide whether there is one pattern or many. If a single pattern is inferred, it simply aggregates all the information, using the method of chapter 4. Otherwise, it applies the method of Chapter 5.

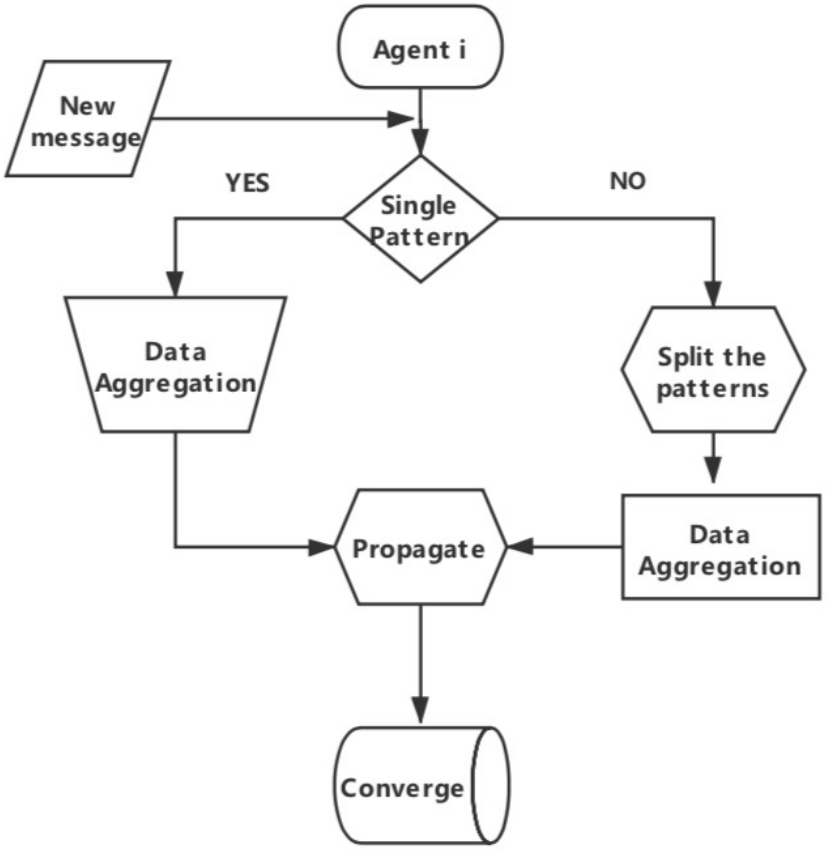


Figure 6.1: Flow chart

That still leaves the question of how determine whether or not the observed data is made up of a single pattern (i.e., all agents are seeing data sampled from the same distribution) or multiple patterns. The basic idea is whether or not there are some agents that are significantly dissimilar to other agents, which is similar to the definition for outlier detection: "given a set of N data points, and n, the expected number of outliers, find the top n points that are considerably dissimilar with respect to the remaining data"[Han et al., 2011]. For outlier

detection technologies, the number of outliers n is fixed in advance. However, we have no idea about the number of outliers, n, or even if there are any outliers.

Given n cluster models $m_1, m_2, \ldots, m_n$, the question is are these cluster models generated from a single distribution or multiple distributions? In this chapter, we consider two different similarity measurements between agents: cluster-level similarity and agent-level similarity. In the following sections, cluster-level similarity based methods are described first, followed by agent-level similarity based methods.

6.2 Cluster-level similarity based methods

Hamerly and Elkan proposed a method G-means to learn the k for k-means [Hamerly and Elkan, 2004]. This hierarchical algorithm assumes that a subset of data that follows a Gaussian distribution is more likely to form a group and a statistical test is used to test whether or not data points belongs to a cluster. It starts with one cluster, then keeps splitting clusters until the points assigned to each cluster have a Gaussian distribution. Here each agent is represented by a set of centroids. Given centroid sets from n different agents, G-means is used to group those centroids into clusters. Any agent is considered from the same pattern if the number of centroids (formed by centroids from different agent models) estimated by G-means is the same as the number of centroids from its agent model.

Figure 6.2 shows an example of detecting single or multiple patterns in a 5-agent network. Obviously, there are more than one pattern in the left figure and only a single pattern in the right figure. Figure 6.3 shows the clusters obtained by G-means using the centroids from different agent models in figure 6.2. On the right side of figure 6.3, there is a single pattern since the number of clusters on centroids from agents' models are the same as the number of centroids before G-means clustering. However, the distance between centroids is ignored. It is possible that centroids that stay close but fit badly to a Gaussian distribution are not assigned to the same cluster.

Ester et al.[Ester et al., 1996] proposed a distance and density based algorithm density-based spatial clustering of applications with noise (DBSCAN) to group points together. Points that are in low-density regions are marked as outliers. This algorithm, which has a low computation cost, is robust to noise and outliers and there is no need to specify the number of desired clusters in advance. However, it requires 2 parameters: the radius of neighbourhood around a point

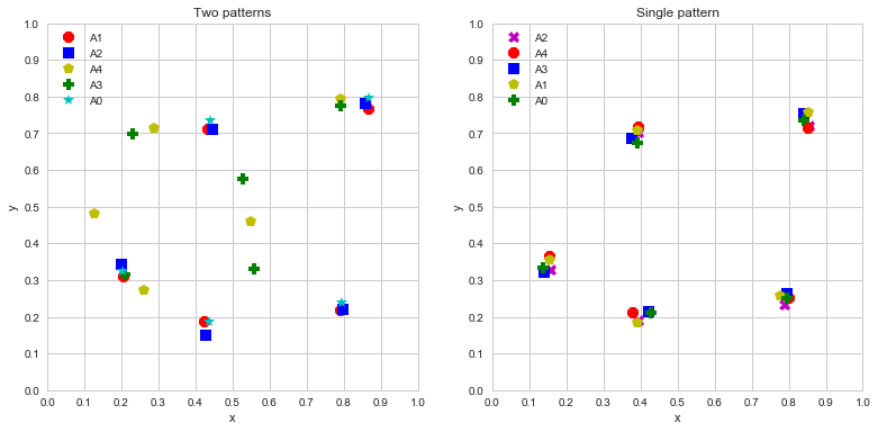


Figure 6.2: One example of detecting single and multiple patterns

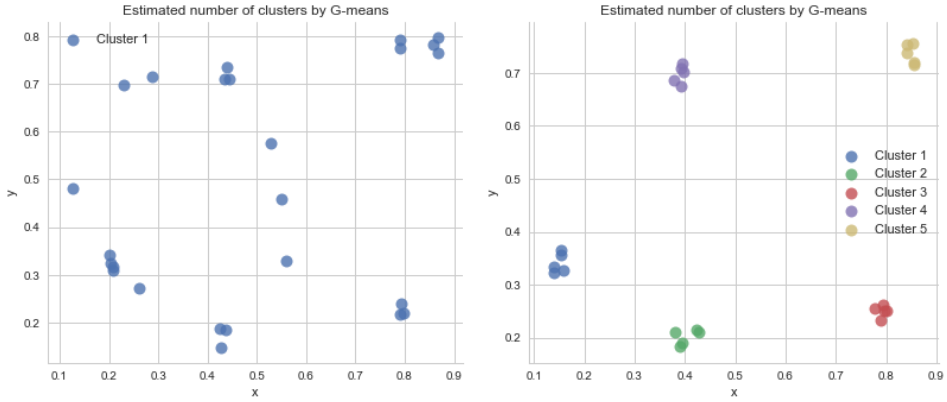


Figure 6.3: Application of G-means to agents' clusters

to define 'neighbours' and the minimum number of points to form a dense region. Even if DBSCAN solved the issue suffered by G-means, selecting the optimal parameters can be difficult since it is very sensitive to any of the two parameters.

Figure 6.4 and 6.5 show that result of pattern detection using the same examples in figure 6.2. Ideally, the coordination of cluster centroids are similar if all agents are sampled from the same distributions. In that case, there are no outliers. If there are multiple patterns, the coordination of cluster centroids are dissimilar, and outliers are detected.

6.3 Agent-level similarity based method

In this section, we describe the agent-level similarity based method. The similarity is calculated based on the representation of agent, such as centroids, size of clusters and shape of clusters. Here distance-based similarity is used to describe the similarity between agents. Figure 6.6 shows an example of similarity matrix

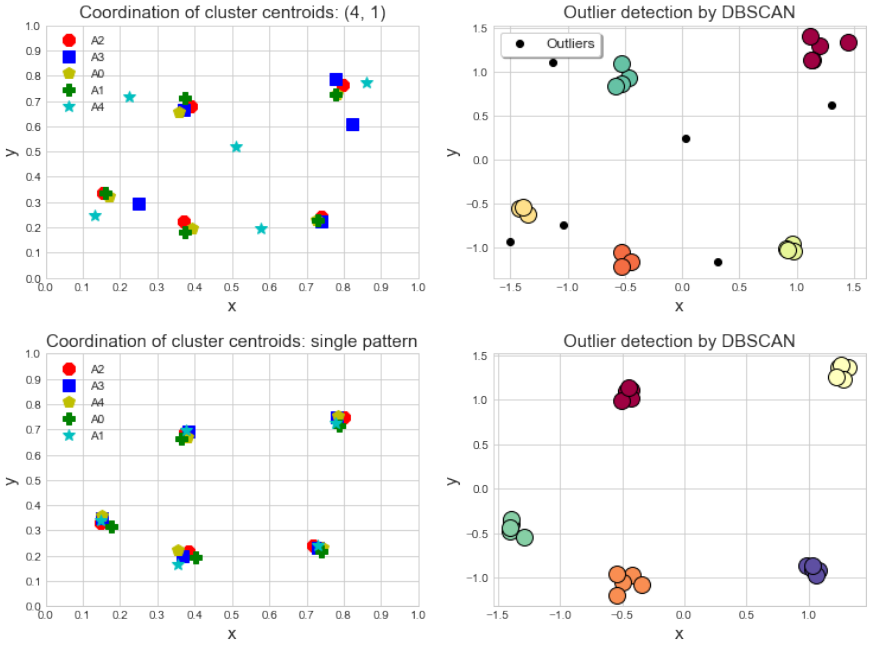


Figure 6.4: Application of DBSCAN to agents' clusters (1)

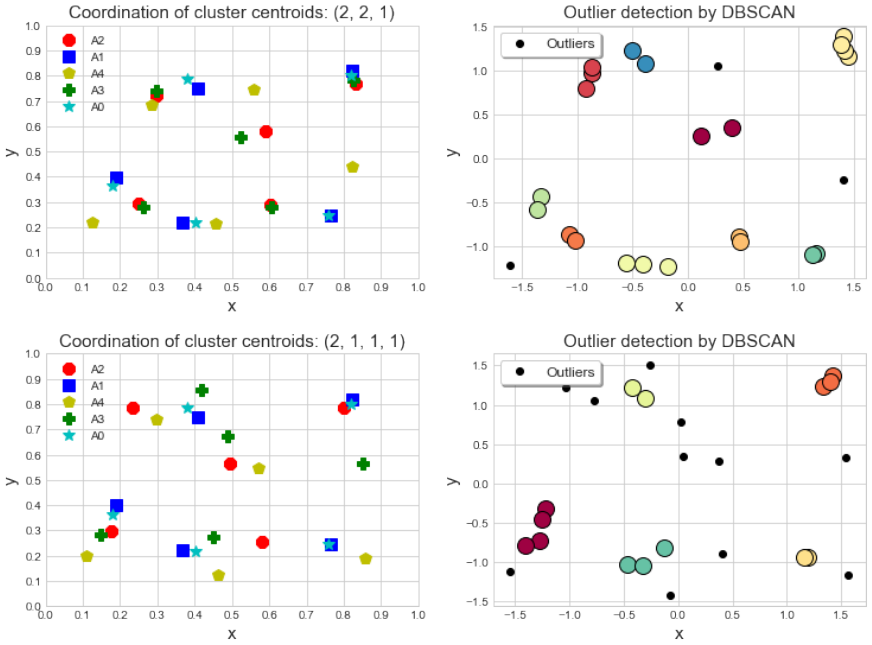


Figure 6.5: Application of DBSCAN to agents' clusters (2)

for a 5-agent network. The distance between an agent to itself is zero. A large value means the two agent descriptions are far apart. For instance, for agent A_1, it is far away from agent A_3 and A_4 but close to A_2 and A_0. For single pattern detection, we use the upper-triangle of this symmetric similarity matrix as the data.

	A1	A2	A4	A3	A0
A1	0.000000	0.021902	0.188831	0.186093	0.021417
A2	0.021902	0.000000	0.187133	0.195990	0.023940
A4	0.188831	0.187133	0.000000	0.136799	0.185735
A3	0.186093	0.195990	0.136799	0.000000	0.183765
A0	0.021417	0.023940	0.185735	0.183765	0.000000

Figure 6.6: Agent-level similarity for a 5-agent network

Taking the symmetric matrix of figure 6.6 as an example, now the question becomes: given a data sequence which represents the euclidean distance based similarity between agents, is it possible to identify whether or not there is more than one pattern?

$$[0.0219, 0.1888, 0.1861, 0.0214, 0.187, 0.1960, 0.0239, 0.1368, 0.1857, 0.1838] \quad (6.1)$$

6.3.1 Kernel Density Estimation

First, we introduce a agent-level similarity method: Kernel Density Estimation (KDE). KDE is a commonly used technique to estimate the probability density function of observations in the data and it is defined as:

$$P_n(x) = \frac{1}{nh} \sum_{i=1} nK(\frac{X_i - x}{h}) \quad (6.2)$$

where $K(x)$ is called the kernel function that is generally a smooth, symmetric function, h is the bandwidth of bins, n is number of points and X_i is the i-th observations. Silverman and Bernard provided a detail review about the density estimation in their work [Silverman, 1986]. However, the choice of bandwidth has a strong influence on the estimation. Small bandwidth causes a large variance while a large bandwidth leads to large bias. In addition, data characteristics, such as sample size, distribution smoothness and skewness, makes it difficult to choose an optimal bandwidth [Heidenreich et al., 2013].

Taking the data sequence 6.1 as an example again, figure 6.7 shows the one dimensional data and its density estimation. Note that the density estimation is based on a bandwidth 0.1. The density curve will vary with different bandwidths. By intuition, there should be two clusters: one cluster is centred around 0.175 and the other cluster is centred around 0.025. The clusters are founded by local minimum of density curve. Figure 6.8 shows the local minimum found (the red dot) with different bandwidths. If the bandwidth is 0.01, there are three clusters: points that located on the left side of 0.08 are grouped as one cluster, points that

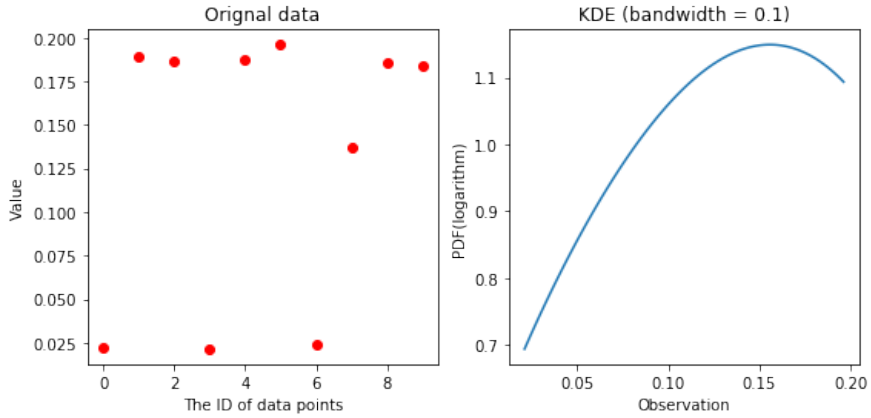


Figure 6.7: Data sequence and its density estimation

located between 0.08 and 0.16 are clustered as one cluster, and the remaining points are grouped into one cluster. When the bandwidth is increased to 0.05, only two clusters are found and those two clusters are divided by 0.08. If the bandwidth is 0.2, only one cluster is found. There is an under-fitting problem if the bandwidth is too small (for instance, bandwidth = 0.01 here) and an over-fitting problem if the bandwidth is too large (for instance, bandwidth = 0.2 here).

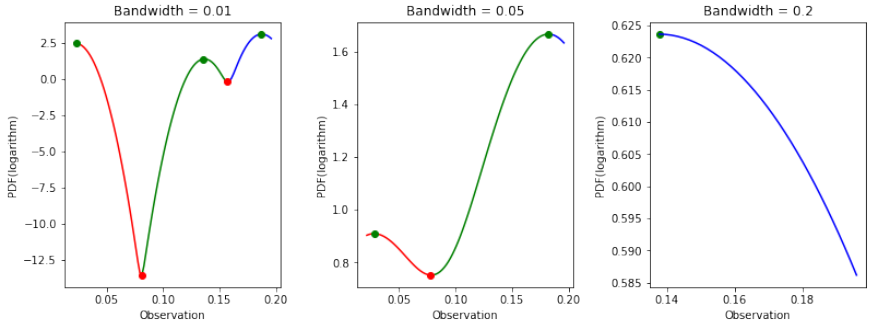


Figure 6.8: Clusters found by KDE

Choosing the optimal bandwidth is widely explored in this area. There are two classes of approaches to this problem. In the statistic community, reference rules are often used to find the optimal bandwidth, such as Silverman's rule and Scott's rule. By Scott's rule, the bandwidth is defined as [Scott, 2015]:

$$B = cAn^{-1/5} \tag{6.3}$$

where c = 1.059, n is the length of data sequence and A is smaller of standard deviation or normalised Interquartile range method (IQR), where IQR is defined as the difference between Q3 and Q1. Silverman's rule assumes Gaussian distri-

bution for the unknown density. Silverman's rule and Scott's rule are normally used as a first estimator since they are computationally fast. However, if the sample size is small, it gives too few bins to mine information from the data.

In the machine learning world, an empirical approach, such as cross validation (CV), is widely used. The procedures for cross validation is as follows: 1), randomly partition the original observations into L equal sized subsets, 2), for each subset, a single subset is retained as test data, and the remaining L-1 subsets are used as training data, and 3) repeat L times, average L results to produce a single estimation. Alternatively, a CV based bandwidth parameter could be adopted from statistics, where the optimal bandwidth could be achieved by maximising the likelihood or minimising the squares. Although, the CV based method is computationally expensive, it is quite useful to avoid the problem of over-fitting.

To address this issue, we proposed selecting the optimal bandwidth via a cross-validation approach such that the mean square distance between the clusters found by KDE is minimal (refer to algorithm 21). For a given search space, do repeated density estimation and the optimal bandwidth is set to be the step with minimum Mean Square Error. We could criticise this method for the repeated process of density estimation, which may increase the overhead for the individual agent. However, only the upper-triangle of similarity matrix is used in our algorithm. Even for a 100-agent network, the data sequence only contains 4,950 one dimensional data, which is an easy job for KDE to handle.

Algorithm 21: Optimised KDE for pattern detection

1 **Require**: Data Sequence D;
Input : Search Space S ($S_n \leq S \leq S_m$), search step S_t
Output: Optimal Bandwidth $\boldsymbol{b}$
2 $\boldsymbol{Sco} \leftarrow \{\}$;
3 **for** *Each* S_t *in* S **do**
4 Fit D with S_t as bandwidth (kernel function is Gaussian);
5 Compute the the Mean Square Error for each cluster;
6 Add Averaged Mean Square Error to $\boldsymbol{Sco}$;
7 $\boldsymbol{b} \leftarrow S_t$ with minimum Averaged Mean Square Error;
8 Return $\boldsymbol{b}$;

Figure 6.9 shows the comparison of various bandwidth estimations with sequence 6.1 as input data. In this example, density estimation by Silverman and Scott bandwidth are much closer to the true PDF curve, followed by our method Estimated bandwidth. However, the results for multiple runs of experiments shows that bandwidth by Silverman and Scott is the worst.

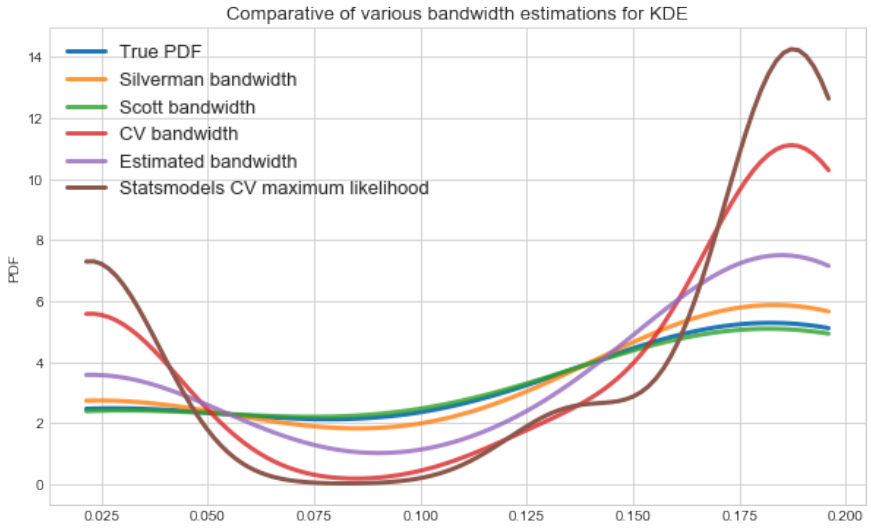


Figure 6.9: Comparative of various bandwidth estimation

6.3.2 Information criterion

In this thesis, we use the AIC to perform model selection. It estimates the relative distance between fitted likelihood function of the model and unknown true likelihood function of the observations, so that a lower AIC means a model is considered to be closer to the truth. When fitting models, it is possible to increase the likelihood by adding parameters, but doing so may result in overfitting. AIC attempts to resolve this problem by introducing a penalty term for the number of parameters in the model.

Note that AIC and KDE make no assumption about the underlying distribution of data points [Anderson and Burnham, 2006]. For AIC, by definition, it is a likelihood function based method to perform model selection and the Gaussian likelihood is most frequently used. There are also some other distributions available, such as Bernoulli distribution, weighted Gaussian distribution and Poisson distribution. By equation 5.4, KDE is a kernel function based method. Generally, a kernel function has two features: 1) $K(x)$ is symmetric, and 2) $\int K(x)dx = 1$. Possible kernel functions include the Gaussian, Uniform and Epanechnikov. Similarly, KDE does not make any assumption about the underlying distribution, although the Gaussian Kernel function is widely used. In addition, the kernel function of KDE does not play a key role in theory, but the selection of bandwidth really matters [Sheather and Jones, 1991].

6.4 Comparison of different methods.

In this section, a comparison of different methods, including DBSCAN, KDE, optimised KDE, G-means and AIC, are described. As before, different clustering algorithms and three different datasets (fully symmetric Gaussians, partially symmetric Gaussian and Uniformly distributed dataset) are used in the experiment. The experiment runs the full algorithm a shown in the flow chart 6.1 and these five methods are used to detecting single or multiple patterns. For each method, it was repeated for 50 runs on a 10-agent dense network to get a more accurate performance.

We consider a 10 agent network (dense or sparse). Each node is given 200 data points and the ground truth number of clusters in all simulations is set to 5. Each message transmission delay is generated uniformly from the range [0.5, 1.0], and nodes begin their work at time randomly selected in [0, 0.1s]. As before, three different underlying 2-dimensional datasets are tested: fully symmetric Gaussians, partially symmetric Gaussians and uniformly distributed datasets. The splits of agents tested are the same as that in Chapter 6: multiple patterns: (5,5), (7,3), (9,1) and (5,3,2) and single pattern (10:0). We are interested in how accurate the algorithms are in predicting whether there is a single pattern or multiple patterns. Note that it is unnecessary to know the exact number of patterns. If multiple patterns have been confirmed, how agents are split will be further estimated by the hierarchical clustering method from Chapter 5.

6.4.1 K-means as clustering algorithm

Figure 6.10 shows the performance when underlying datasets are generated from fully symmetric Gaussians and K-means is used as the clustering algorithm. Optimised KDE and AIC perform better than other methods in predicting there are multiple patterns, and the accuracy is around 98%. When the ground truth is a single pattern, G-means outperforms other methods, and the accuracy to predict it is a single pattern is up to 78%. When the number of clusters over patterns is different, there is little difference between accuracies of predictions.

When the underlying datasets are generated from partially symmetric Gaussians, Optimised KDE and DBSCAN achieve higher accuracy than other methods in predicting there are multiple patterns (around 97% on average). In term of ground truth single pattern, KDE outperforms other methods. The accuracy achieved by KDE is about 58%, nearly 3 times as much as that achieved by DBSCAN and

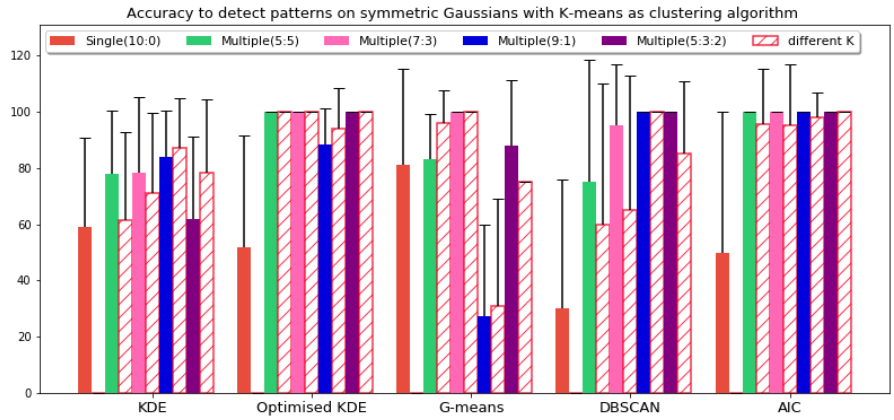


Figure 6.10: Accuracy achieved with symmetric Gaussians when K-means is used

G-means. The performance is improved noticeably when the number of clusters over patterns are different.

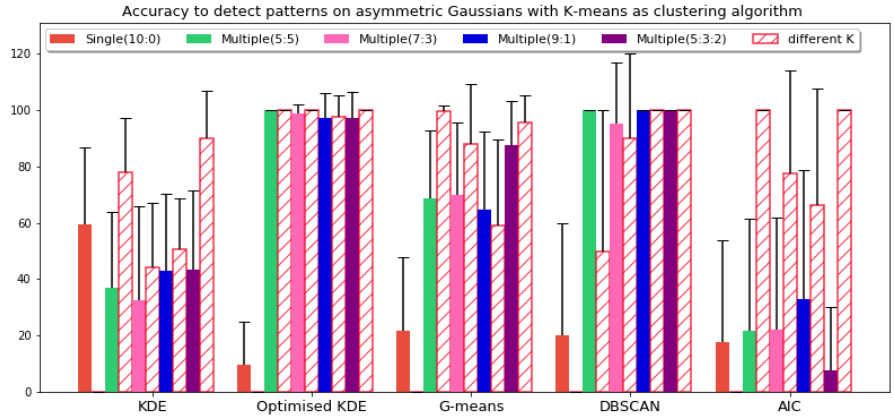


Figure 6.11: Accuracy achieved with asymmetric Gaussians when K-means is used

Figure 6.12 shows the performance when underlying datasets are uniformly distributed. The highest accuracy in predicting there are multiple patterns is obtained by optimised KDE and DBSCAN. However, they perform the worst in detecting there is a single pattern. The highest accuracy in predicting the single pattern is obtained by AIC (around 65%), followed by KDE (about 52%).

6.4.2 GMM as clustering algorithm

In this section, we show the performance when GMM is applied as clustering algorithm. Figure 6.13 shows the performance when underlying datasets are

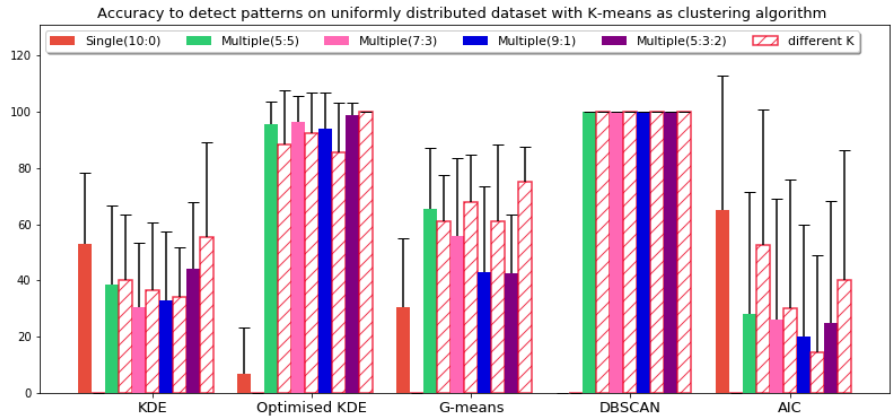


Figure 6.12: Accuracy achieved with uniformly distributed dataset when K-means is used

generated from fully symmetric Gaussians. As before, the results are showing the same behaviour.

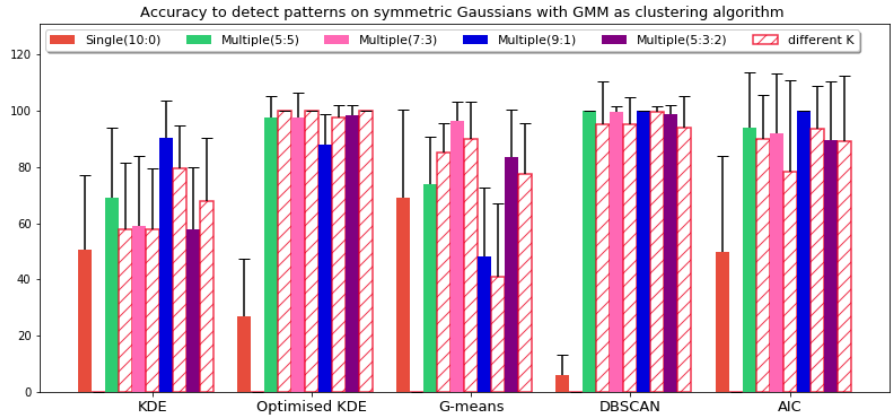


Figure 6.13: Accuracy achieved with symmetric Gaussians when GMM is used

When datasets sampled from partially symmetric Gaussians are used as underlying dataset (refer to figure 6.14), the result is similar to that with K-means as the clustering algorithm.

Figure 6.15 shows that G-means and Optimised KDE are likely to predict there are multi patterns even when the ground truth is a single pattern while KDE and AIC are prone to predicting there is single pattern even when ground truth has multiple patterns when underlying dataset are sampled from uniform distributions.

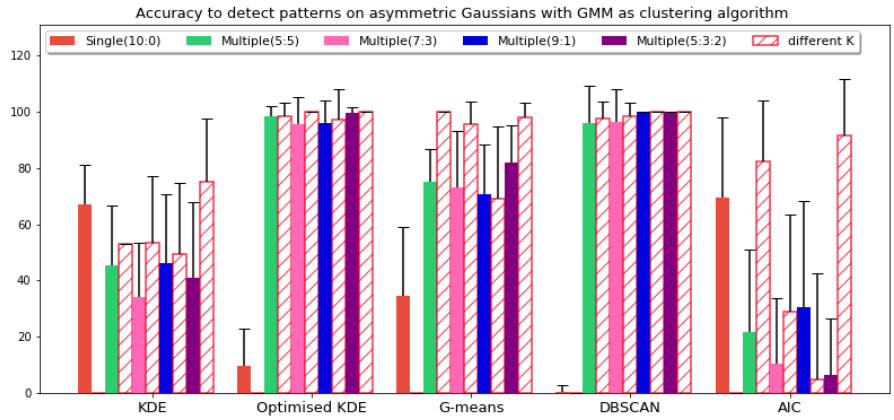


Figure 6.14: Accuracy achieved with asymmetric Gaussians when GMM is used

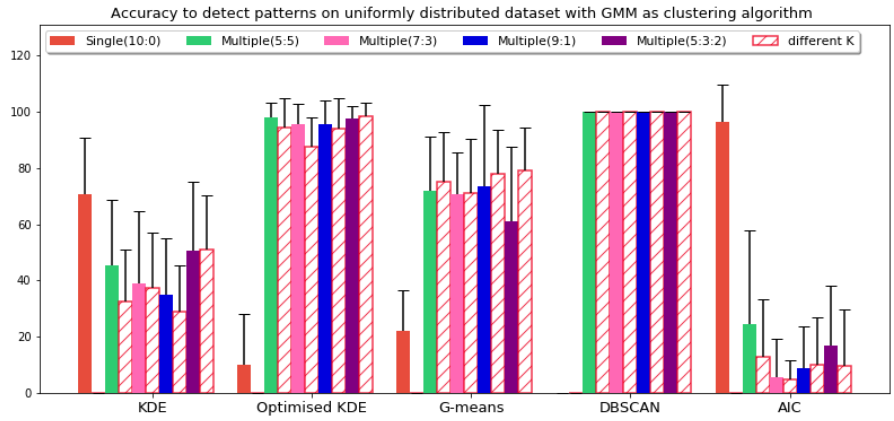


Figure 6.15: Accuracy achieved with uniformly distributed dataset when GMM is used

6.4.3 SG as clustering algorithm

When SG is used as the clustering algorithm, Optimised KDE, DBSCAN and AIC achieve a really high accuracy to detect multiple patterns when the underlying datasets are generated from fully symmetric Gaussians. The highest accuracy in detecting the single pattern is achieved by G-means. However, it performs worse in detecting multiple patterns.

Figure 6.17 shows the performance when datasets generated from partially symmetric Gaussians are used. The results show the same behaviour as that with K-means as the clustering algorithm.

When datasets are sampled from uniform distributions (refer to figure 6.18), the results are similar to that with GMM as clustering algorithm.

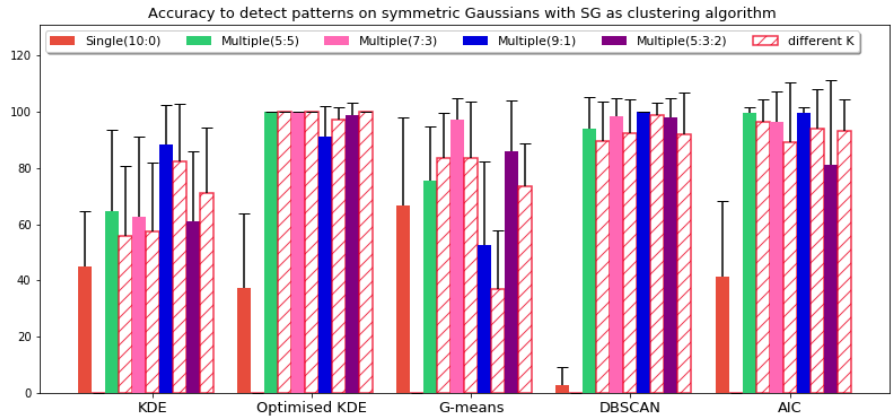


Figure 6.16: Accuracy achieved with symmetric Gaussians when SG is used

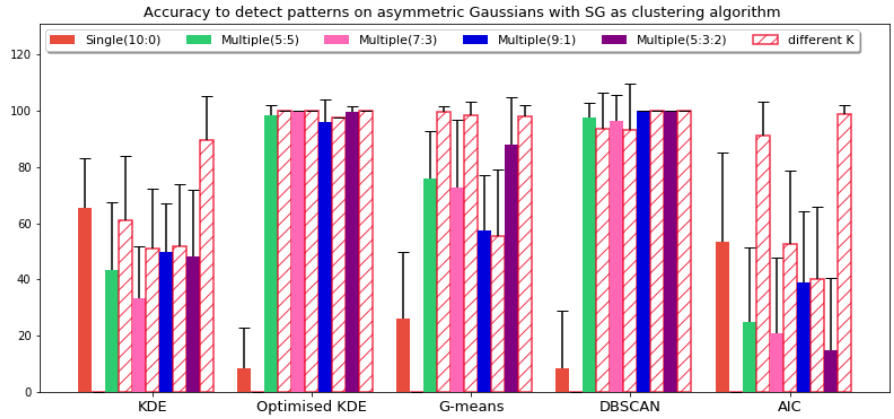


Figure 6.17: Accuracy achieved with asymmetric Gaussians when SG is used

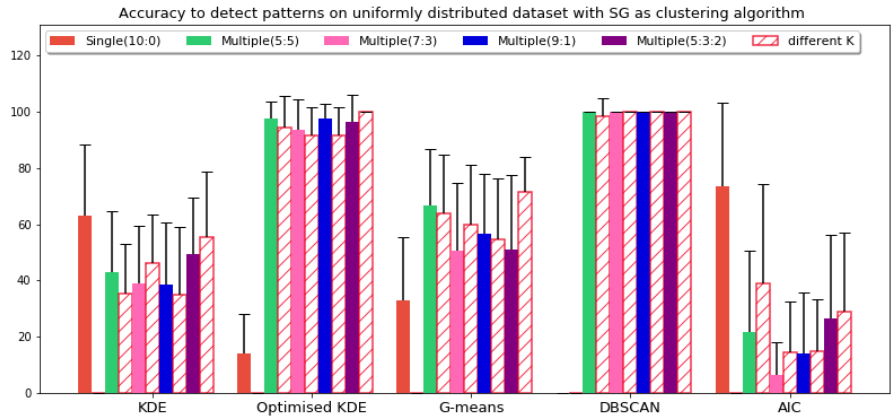


Figure 6.18: Accuracy achieved with uniformly distributed dataset when SG is used

6.4.4 Summary

Overall, Optimised KDE and DBSCAN performs better in detecting multiple patterns, and AIC and KDE outperforms other methods in predicting single pattern.

Note that the accuracy against data points could be as high as that in single pattern even when multiple patterns are predicted. It happens when the similarity between agents is not so distinct. We will consider accuracy against data points in the next section.

6.5 Accuracy against data point with different predictions

At the end of the last section, we noted that a poor prediction of single or multiple patterns might not affect accuracy against data clustering, if the agents in different patterns are still similar. In this section, we explore this, and evaluate accuracy against data points. Since optimised KDE and DBSCAN perform well in detecting multiple patterns, we compared the accuracy against data points by these two methods. In previous chapters, we used two methods to measure the accuracy against data points: one is fitting the learned model to agent's local data points (denoted by m1), the other one is fitting the learned model to a new test data obtained by sampling from the same ground truth distribution (denoted by m2). The performance of these two methods is measured over 20 runs.

Figure 6.19 shows the accuracy against data points achieved by optimised KDE when the ground truth has a single pattern. Right prediction covers the cases where the algorithm predicts the pattern is single, and the ground truth pattern is single. Wrong prediction covers the cases where the algorithm predicts there are multiple patterns. The x ticks show the three different clustering algorithms applied to three different underlying datasets. For instance, Kmeans(s) shows that k-means algorithm is used to fit the datasets generated from fully symmetric Gaussians.

Comparing red bars with green bars, we can see there is little difference between the right prediction and wrong prediction. The blue and pink bars show that the accuracy m2 by wrong predictions is slightly higher than that by right prediction. Note that we have shown the accuracy to detect the single pattern is quite low in previous chapter. Taking asymmetric Gaussians as an example, the accuracy was

less than 10%. Over the 20 runs, it only predicts a single pattern twice. That is the reason why accuracy against data points is slightly higher even when the prediction is wrong.

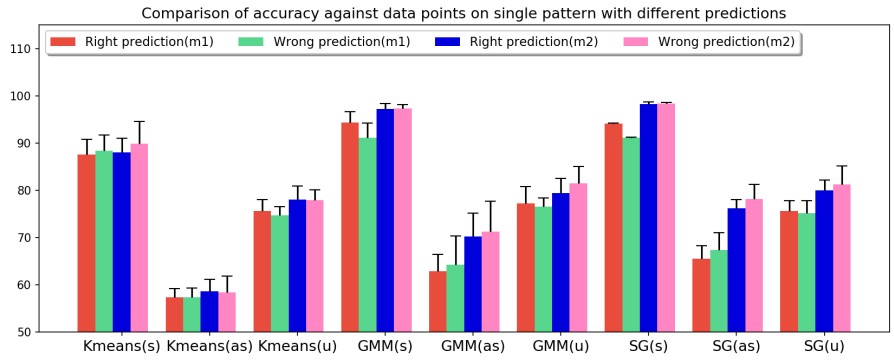


Figure 6.19: Accuracy against data points achieved by Optimised KDE when ground truth pattern is single

Figure 6.20 shows the accuracy against data points achieved by DBSCAN. There is little difference over accuracy m1 and m2 between different predictions. In some cases, the accuracy achieved by wrong prediction is a little higher than the right prediction. It indicates that the accuracy against data remains stable even when the prediction is wrong and the ground truth single pattern is divided into sub-patterns.

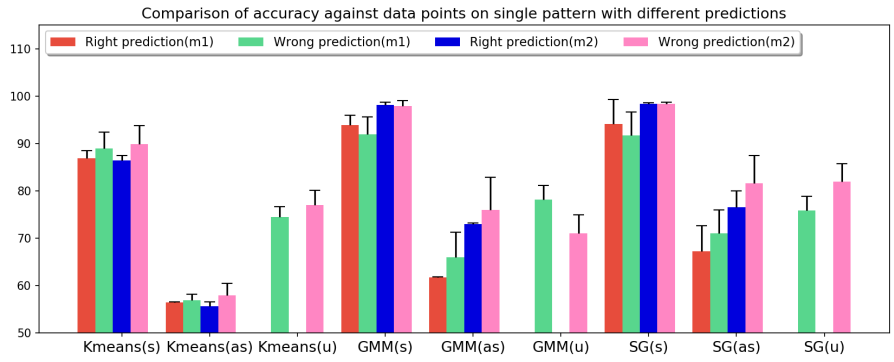


Figure 6.20: Accuracy against data points achieved by DBSCAN when ground truth pattern is single

To summarise, the accuracy against data points remains stable even when the ground truth pattern is single and it is divided into multiple patterns. We have shown that predicting the correct number of patterns is crucial for high accuracy against pattern in previous chapter. Since the prediction of number of patterns is wrong, we expected that the accuracy against pattern is low.

6.6 Conclusion

Overall, KDE and AIC outperform other methods in detecting single pattern, but they perform worse in detecting multiple patterns. Optimised KDE and DBSCAN perform better than other methods in detecting multiple patterns, but they are not good at detecting a single pattern. Which method should be used to detect the number of patterns depends on the frequency of pattern occurrence. Optimised KDE and DBSCAN should be considered if there is a high frequency of multiple patterns.

Note that it is challenging to detect that there is only a single pattern. The highest accuracy to detect single pattern is only around 65%. However, the proposed methods Optimised KDE and DBSCAN perform well in detecting multiple patterns, and the highest accuracy to detect multiple patterns is nearly 99%. Based on the basic procedure of algorithm, we expect the accuracy against patterns to drop but accuracy against data points to remain the same even if the single pattern is divided into sub-patterns. Experimental results show that there is little difference in accuracy m1 and m2 against data points between right prediction and wrong prediction. It indicates that our algorithm could always predict how data points are clustered even if there are multiple patterns. But the prediction of sub-communities is not reliable, since it is likely that it divides one big community into two or more similar communities. A further comparison of those sub-communities is needed if the detection of communities really matters.

Chapter 7

Conclusions and Future Work

7.0.1 Conclusions

In this thesis, we consider a problem motivated by Wireless Mesh Networks: how should the agents (or nodes in the network) act and communicate so that all agents end up with the same understanding of the environment? Transmitting all data to the centre allows strong data analytics algorithms to be applied, but consumes battery power for the nodes, may cause network congestion, and may breach the privacy of the original data. To avoid this, distributed methods try to learn within the network, allowing each sensor to learn a global picture and take appropriate actions. The hypothesis defended in this thesis is: distributed clustering in wireless mesh networks is possible by using an asynchronous adaptive clustering algorithm without transmitting raw data, and without incurring high latency or high energy costs, but can achieve high clustering quality and even when extended to scenarios where there are sub-groups of agents are receiving data from different environments.

Firstly, we assume that all agents are receiving data from the same environment (i.e from the same data distribution) and the number of clusters is known by agents in advance. We consider two levels of agents' knowledge: agents know the number of nodes in this network and agents know their direct neighbours only. We also consider two different underlying clustering algorithms: K-means and GMM. A bounding box based method is proposed to describe the shape information of clusters if K-means is used as the clustering algorithm. Experimental results show that the algorithms, which exchange cluster shape and density descriptions, require fewer messages and give higher accuracy relative to centralised methods. Compared to the state of the art, our proposed algorithms reduce the

misclassification rate from 12% to 1.7%, reduce the message count by 20%, and reduce elapsed time by 75%.

Then, we relax the assumptions that the number of clusters is known by agents in advance. We consider two new assumptions about the network: either that all agents may finish with slightly different models or all agents must finish with identical models. An agent estimates the number of clusters locally, then use the silhouette method to infer the number of clusters from the re-generated data. Two centralising methods, centralising raw data points to a designated central agent or transmitting a basic model to the designated central agent, are also considered. Experimental results show that the extent of overlapping between clusters has a big effect on the learned number of clusters. When clusters are well separated from each other, all proposed methods learn the same number of clusters as ground truth. For distributed methods, more information is required to be transmitted if it is required that agents must finish with identical models.

After that, we extend our algorithm to scenarios where there are sub-group of agents are receiving data from different environments. The aim is now to identify all of these patterns and to associate each agent with the appropriate pattern. We assume that there is more than one pattern. A weighted Earth Mover's Distance (EMD) is proposed to measure the similarity or difference between two cluster models. We can then use this to search for clusters of similar agents. An additional measurement accuracy against pattern, which measures how accurately the agents are assigned into groups or patterns compared to the original ground truth, is proposed. We also consider two new requirements for communicating the final inferred model: each agent is told the number of patterns and summary description of each pattern only or each agent is told the number of patterns, the pattern description and the details of assignment of all agents. The experiments show that the proposed algorithm performs well when agent models are dissimilar, or the number of clusters are the same and the number of clusters for agents over patterns is different.

Finally, we tackle the problem of how to distinguish an environment with a single pattern and one with multiple patterns. Two types of methods are proposed: cluster-level and agent-level. Experimental results show that KDE and AIC outperform other methods in detecting single pattern, but they perform worse in detecting multiple patterns. Optimised KDE and DBSCAN perform better than other methods in detecting multiple patterns, but they are not good at detecting a single pattern. Note that the accuracy against data points remains stable

even when the ground truth pattern is single and it is divided into multiple patterns. Which method should be used to detect the number of patterns depends on the frequency of pattern occurrence. Optimised KDE and DBSCAN should be considered if there is a high frequency of multiple patterns.

Overall, we proposed asynchronous distributed clustering algorithms, which do not require the number of clusters as input, nor did they need to know the number of pattens in advance. Two levels of agent's knowledge, two assumptions of final models, and two requirements of sharing strategies are considered. Experimental results show that the proposed algorithms can learn how data points are clustered even when there are multiple patterns in the network. In addition, the estimated number of clusters is close to the ground truth clusters.

7.0.2 Future work

Note that there are some assumptions about the algorithms. Firstly, the underlying network is connected and agents are only allowed to share message with their neighbours. Secondly, the network ensures reliable delivery of message, but there is a message-specific delay. Finally, all agents initialise their work at a pre-scheduled time. However, the process of message transmission in the network is complex. A potential further direction would be testing the algorithms with some real testbeds, for instance, NS3. The convergence time measured by NS3 will consider both the real transmission time and computation cost for processing the raw data.

First, we will extend the evaluation, to consider larger networks and different data distributions. In this thesis, we considered data points that generated from fully symmetric Gaussians, partially symmetric Gaussians and rectangle uniform distributions. In some practical applications, there are some other distributions. For different distributions, an appropriate clustering algorithm and efficient method to describe the shape of clusters are needed. We will extend algorithms to scenarios that the network topology changes over time. Some agents may turn to sleep or turn down for lacking of battery. In that case, agent, who is running out battery, has to inform all the neighbours. Neighbours will update their knowledge about immediate neighbours and stop forwarding message to this agent.

We will extend our approaches to more than 2-dimensional spatial datasets. We will address application scenarios where the distributions change over time. In addition, we will extend our methods to handle different inference problems, identifying which problems can or should be handled in the network, and which

require transmission to a central server for more extensive analysis.

Chapter 8

Appendices

8.1 Evaluation of information exchange method

In this section, we evaluate the performance of method proposed to describe the clusters. The performance is measured by comparing the summary description of generated data points with the ground truth summary description. To measure how close the new summary description is to the ground truth summary description, we calculate the difference of three measurements before and after regeneration:

1. Euclidean Distance between the original centroids and new centroids,
2. Shift distance of bounding box (measures the shift distance of four coordinates (leftmost, rightmost, bottom, top) of the box), and
3. Rotation degree from old PCs to new PCs.

Actually, the arrows plotted in figure 3.3 should be two-way arrows, which means we could rotate the PCs from both direction. We define the rotation degree as the minimum rotate degree of the four directions of two PCs to remap new PCs from old PCs. For each case, data points are generated from fixed distributions and different distributions. The results shown below are the averaged values over multiple runs.

8.1.1 Measurement with Dataset Generated on Standard Basis

First, we consider the scenario that data points are generated from one binormal distribution ($N(-0.2, Sig_1)$ and $N(0, Sig_2)$. The dataset is scaled to [0,1].

Table 8.1 shows the measurement on over 20 runs, for different values of the variances. Note that the mean in the table is absolute value, including mean absolute distance, mean absolute rotation. Compared with the original centroids and bounding box, the generated centroids and bounding boxes has slightly changed. The centroids move by only a small distance. The shift distance between original centroids and generated centroids is below 0.06 with a small standard deviation (around 0.03). The shift distance for the three bounding boxes are around 0.09 on average. The bounding boxes are also stable, although the outer boxes are moving a little further, particularly as the cluster shapes moves away from being circular.

In term of rotation degree, the worst average rotation is less than 2 degree, which

indicates that the new PCs are quite faithful to the original PCs. The re-created PCs are only a small rotation away from the original PCs when there is a large difference in the variance of two axis. For example, when the difference is 0.5, it only requires to rotate 0.23 degree over multiple runs with a small standard deviation 0.86. But it requires to rotate 1.89 degree to remap the new PCs when the range is decreased to 0.1.

When the two variances of the binormal are similar, the standard deviation is higher (over 10), and so sometimes the PCs may have rotated by a noticeable amount. This result could be expected. A minimal difference in the variances means the original cluster was approximately circular, and so small changes in the distribution points could have a significant effect on the principal component; a large difference means the cluster was elliptical, and small changes to the data points are less likely to change the PC.

Table 8.1: Measurement for fixed distributions with standard basis

Data	Measures	Mean	S.D
N(-0.2,0.6),N(0,0.1)	Distance between centroids	0.057	0.031
	Rotation degree	0.23	0.86
	Distance between 100% box	[0.06, 0.112,0.059, 0.061]	0.037
Estimated variance:	Distance between 80% box	[0.096,0.09,0.06,0.064]	0.055
(0.685, 0.108)	Distance between 40% box	[0.07, 0.061, 0.058, 0.06]	0.04
N(-0.2,0.5),N(0,0.2)	Distance between centroids	0.059	0.026
	Rotation degree	0.40	2.55
	Distance between 100% box	[0.06, 0.081,0.059, 0.062]	0.044
Estimated variance:	Distance between 80% box	[0.096,0.106,0.073,0.065]	0.065
(0.538, 0.212)	Distance between 40% box	[0.09, 0.059, 0.066, 0.061]	0.034
N(-0.2,0.4),N(0,0.3)	Distance between centroids	0.042	0.029
	Rotation degree	1.89	11.67
	Distance between 100% box	[0.043, 0.071,0.044, 0.063]	0.04
Estimated variance:	Distance between 80% box	[0.052,0.115,0.077,0.053]	0.04
(0.421, 0.311)	Distance between 40% box	[0.047, 0.055, 0.058, 0.046]	0.03

Then dataset generated from different distributions is evaluated, where both the mean and standard deviation are varied. Table 8.2 shows the result. We get a similar result as the previous case. Again, the shift distance of centroids and bounding boxes have slightly changed. The shift distance of boxes increase when the range of variance decrease. Meanwhile, the rotation degree required to remap new PCs increase when the range of variance decrease.

To conclude, the summary descriptions do appear to be stable after data regeneration, changing the coordinates, shape and rotation by only small amounts. Descriptions for clusters that are elliptical are more stable than those for circular clusters.

Table 8.2: Measurement for different distributions with standard basis

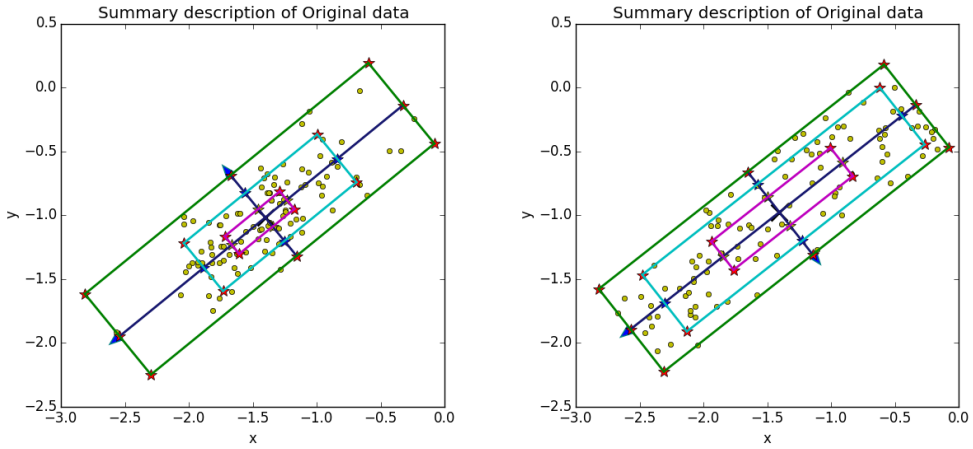
Standard deviation	Measures	Mean	S.D
Sig_1= 0.6,Sig_2= 0.1	Distance between centroids	0.063	0.043
	Rotation degree	0.175	0.87
	Distance between 100% boxes	[0.117, 0.099,0.076, 0.074]	0.062
	Distance between 80% boxes	[0.087,0.085,0.047,0.043]	0.046
	Distance between 40% boxes	[[0.086, 0.063, 0.075, 0.072]	0.052
Sig_1= 0.5,Sig_2= 0.2	Distance between centroids	0.044	0.031
	Rotation degree	0.76	2.85
	Distance between 100% boxes	[0.101, 0.137,0.081, 0.071]	0.065
	Distance between 80% boxes	[0.114,0.14,0.081,0.079]	0.07
	Distance between 40% boxes	[[0.077, 0.071, 0.068, 0.071]	0.041
Sig_1= 0.4,Sig_2= 0.3	Distance between centroids	0.050	0.031
	Rotation degree	3.51	11.71
	Distance between 100% boxes	[0.191, 0.327,0.17, 0.258]	0.21
	Distance between 80% boxes	[0.177, 0.251, 0.165, 0.22]	0.18
	Distance between 40% boxes	[[0.187, 0.205, 0.179, 0.205]	0.14

8.1.2 Measurement with Different Number of Data Points

In this section, we use different number of data points to test the performance. Data points are generated from different distributions but with a fixed standard deviation (0.5, 0.2). Table 8.3 shows that the shift distance between centroids increase when the number of data points decrease from 200 to 50. But for N= 50, the shift distance is still low. The rotation is low and the boxes have moved, though, so 50 is close to the lower limit. Note that although the mean of rotation is low for N=50, the standard deviation is higher, and increases as N decreases.

Table 8.3: Measures varied with number of data points

N	Measures	Mean	S.D
50	Distance between centroids	0.071	0.039
	Rotation degree	-0.014	4.468
	Distance betweenbounding box	[0.12, 0.243, 0.114, 0.14]	0.116
	Distance between 80% box	[0.129, 0.189, 0.121, 0.126]	0.091
	Distance between 40% box	[0.121, 0.139, 0.137, 0.121]	0.077
100	Distance between centroids	0.051	0.026
	Rotation degree	0.06	2.98
	Distance between bounding box	[0.076, 0.144, 0.075, 0.086]]	0.066
	Distance between 80% box	[0.126, 0.11, 0.082, 0.076]	0.087
	Distance between 40% box	[0.08, 0.076, 0.071, 0.07]	0.049
200	Distance between centroids	0.039	0.020
	Rotation degree	0.78	1.98
	Distance between bounding box	[0.079, 0.099, 0.065, 0.057]	0.049
	Distance between 80% box	[0.098, 0.087, 0.075, 0.053]	0.048
	Distance between 40% box	[0.065, 0.06, 0.06, 0.057]	0.032



(a) Original data (Normal)] (b) Original data (Uniform)

Figure 8.1: Data generated by normal and uniform in same range

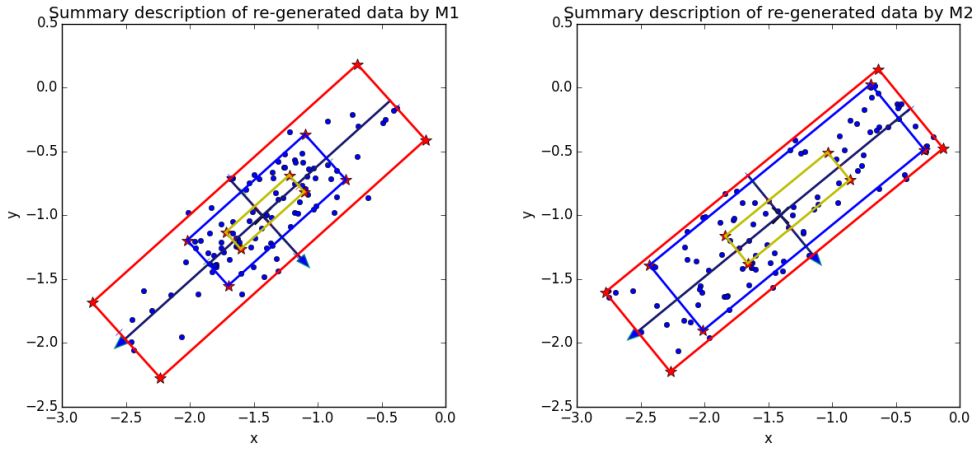
8.1.3 Measurement with Different Underlying Distributions

Previously, data points generated from normal distribution are considered, and in this section data points generated from uniform distributions are considered as well. To ensure that the comparison is fair, data points generated from normal or uniform distributions should be located in same range. Each time a set of data is generated from normal distributions, the 100% bounding box are saved, and it is used to generate another set of data uniformly. Figure 8.1a shows that data created from normal distributions and its 100% bounding box. Based on the green bounding box, another set of data was generated uniformly(refer to figure 8.1b), so both generated data are ready to compare.

Following the steps we have done in the pervious section, the next step is generating new set of data based on the summary description of created original data. Here we consider two different methods to generate data points here:

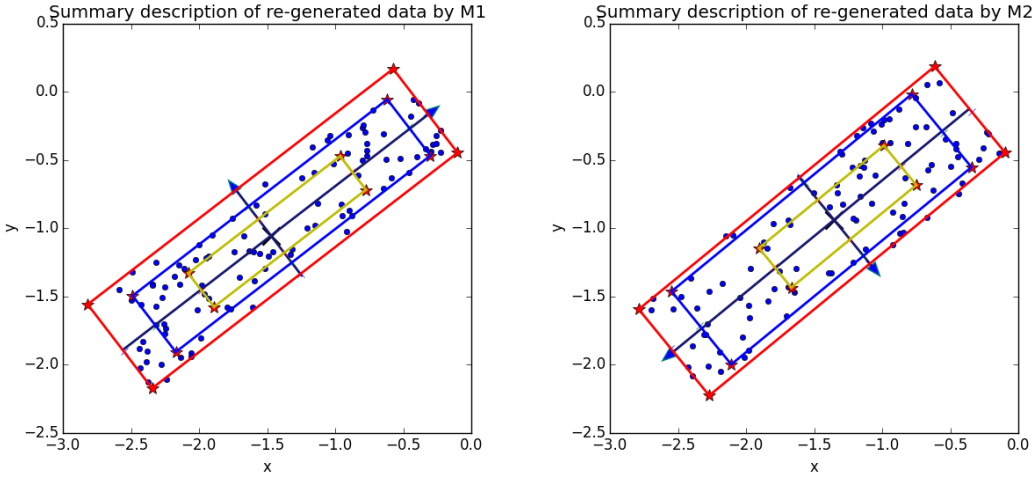
1. Randomly generate an x coordinate and find its right place by comparing with the percentiles, then generate a random number in the corresponding place of the percentile value. Data points are uniformly generated in 5 * 5 cells. This method is denoted by M1.

2. The second method is quite similar to the M1 but data points are randomly generated in only one cell (the 100% bounding box), and it was denoted by M2.



(a) re-generated data by M1 (Normal)] (b) re-generated data by M2 (Normal)

Figure 8.2: Data re-generated by M1 and M2 (Normal)



(a) re-generated data by M1 (uniform)] (b) re-generated data by M2 (uniform)

Figure 8.3: Data re-generated by M1 and M2 (uniform)

Figure 8.2a and 8.2b show the re-generated data by method M1 and M2 separately according to the summary description of data points created from normal distributions in figure 8.1a. It shows that the re-generated data by M1 is similar to the created data. Figure 8.3a and 8.3b show the re-generated data by method M1 and M2 separately according to the summary description of data points created from uniform distributions in figure 8.1b. Results shows that there is a little difference between the data generated by M1 and M2.

Table 8.4 shows that shift distance of bounding boxes and centroids of data points

generated by normal distributions is larger than that by uniform distributions. However, the standard deviation of rotation degree is much smaller. Compared normal (M1) with normal (M2), we can see that the M1 performs better than M2, it makes senses since more information are provided. However, there is a little difference between uniform (M2) and uniform (M1). It makes senses since we consider the fact that here is a little difference between randomly generate data points in a big box and randomly generate data points in multiple sub-boxes when the underlying dataset is uniform.

Table 8.4: Measures varied with underlying distributions

Distributions	Measures	Mean	S.D
Normal (M1)	Distance between centroids	0.047	0.029
	Rotation degree	0.03	2.559
	Distance between bounding box	[0.078, 0.089, 0.061, 0.062]	0.048
	Distance between 80% box	[0.091, 0.109, 0.072, 0.06]	0.051
	Distance between40% box	[0.068, 0.078, 0.052, 0.048]	0.037
Normal (M2)	Distance between centroids	0.137	0.106
	Rotation degree	1.67	2.96
	Distance between bounding box	[0.055, 0.053, 0.123, 0.123]	0.073
	Distance between 80% box	[0.262, 0.482, 0.199, 0.204]	0.131
	Distance between 40% box	[0.207, 0.388, 0.178, 0.171]	0.111
Uniform (M1)	Distance between centroids	0.045	0.027
	Rotation degree	0.100	4.270
	Distance between bounding box	[0.038, 0.04, 0.036, 0.035]]	0.025
	Distance between 80% box	[0.055, 0.075, 0.04, 0.047]	0.036
	Distance between 40% box	[0.075, 0.078, 0.056, 0.06]	0.042
Uniform (M2)	Distance between centroids	0.095	0.064
	Rotation degree	0.668	4.656
	Distance between bounding box	[0.036, 0.043, 0.086, 0.086]]	0.047
	Distance between 80% box	[0.095, 0.08, 0.104, 0.098]	0.067
	Distance between 40% box	[0.125, 0.136, 0.1, 0.115]	0.078

8.1.4 Measurement with Different Percentiles

In this section, we investigate how different percentiles for the bounding boxes affect the measures. Data points are generated from different distribution but with a fixed standard deviation (0.5,0.2).

Table 8.5 shows that performance achieved by different percentiles. Comparing the first four experiments, the overall performance achieved by percentile arrays [0.0,0.1,0.3 0.7,0.9,1.0] and [0.0,0.1,0.3,0.7,0.9,1.0] are a little better. The rotation degree by [0.0,0.2,0.4,0.6,0.8,1.0] is higher and the standard deviation is higher as well. Comparing the shift distance of the 100% bounding boxes, we could know shift distance for [0.0,0.05,0.3,0.7,0.95,1.0] is smaller than the others, followed by [0.0,0.1,0.3,0.7,0.9,1.0]. However, the rotation degree required is higher. The

most likely reason lies in percentile value we choose. The PCs are mainly effected by the boundary points, which located inside the 100% bounding box but outside the second inner bounding boxes.

In the later two rows, we increase the number of bounding boxes. We see that the rotation degree, shift distance of centroids, shift distance of bounding boxes are worse when we increase the number. There appears to be little advantage in generating more bounding boxes in the summaries.

Table 8.5: Measures varied with percentiles

Percentile	Measures	Mean	S.D
[0.0,0.1,0.3 0.7,0.9,1.0]	Shift Distance between centroids	0.057	0.03
	Rotation degree	-0.472	2.67
	Shift distance of bounding box	[0.073, 0.096, 0.063, 0.065]]	0.043
	Shift distance of 80% box	[0.099, 0.092, 0.071, 0.073]	0.06
	Shift distance of 40% box	[0.079, 0.08, 0.065, 0.062]	0.043
[0.0,0.05,0.3 0.7,0.95,1.0]	Shift Distance between centroids	0.054	0.03
	Rotation degree	-0.692	3.37
	Shift distance of bounding box	[0.048, 0.077, 0.049, 0.051]]	0.044
	Shift distance of 90% box	[0.184, 0.231, 0.11, 0.084]	0.082
	Shift distance of 40% box	[0.078, 0.067, 0.066, 0.06]	0.04
[0.0,0.2,0.4, 0.6,0.8,1.0]	Shift Distance between centroids	0.038	0.022
	Rotation degree	0.329	2.72
	Shift distance of bounding box	[0.08, 0.096, 0.051, 0.049]	0.064
	Shift distance of 60% box	[0.162, 0.157, 0.061, 0.078]	0.042
	Shift distance of 20% box	[0.074, 0.061, 0.041, 0.044]	0.025
[0.0,0.3,0.4, 0.6,0.7,1.0]	Shift Distance between centroids	0.042	0.019
	Rotation degree	1.02	3.87
	Shift distance of bounding box	[0.092, 0.082, 0.054, 0.061]	0.054
	Shift distance of 40% box	[0.113, 0.166, 0.068, 0.064]	0.03
	Shift distance of 20% box	[0.052, 0.071, 0.047, 0.04]	0.023
[0.0,0.1,0.3,0.4 0.6,0.7,0.9,1.0]	Shift Distance between centroids	0.217	0.032
	Rotation degree	0.519	3.51
	Shift distance of bounding box	[0.091, 0.179, 0.203, 0.21]]	0.054
	Shift distance of 80% box	[0.15, 0.445, 0.204, 0.279]	0.072
	Shift distance of 40% box	[0.1, 0.385, 0.204, 0.25]	0.037
[0.0,0.1,0.2,0.4 0.6,0.8,0.9,1.0]	Shift Distance between centroids	0.259	0.046
	Rotation degree	-0.43	2.69
	Shift distance of bounding box	[0.119, 0.148, 0.24, 0.245]]	0.061
	Shift distance of 80% box	[0.128, 0.438, 0.245, 0.3]	0.079
	Shift distance of 60% box	[0.144, 0.447, 0.244, 0.294]	0.046

8.1.5 Possible issue caused by regeneration

Based on the previous sections, regenerating data from the bounding box descriptions does seem to be feasible, and would allow an agent to generate similar clusters of data. But there may be an issue if this is done repeatedly. If a bounding box extends to exactly the bounds of the extreme points, when we regenerate

points inside the box, we are unlikely to generate points on the edges, and so the spread of the cluster shrinks after each regeneration, and repeated regeneration will make it worse.

Based on the received summary description of k clusters, agent can recover the data. However, there is a shrink of generated data compared to the original data. In order to address this issue, we considered four potential solutions:

1. Do nothing
2. Impose 4 data points on the extremes of the outer box during regeneration
3. Adjust the bounding boxes on creation by expanding the inner boxes to midway to the next relevant point
4. Combine step 3 and 4 above

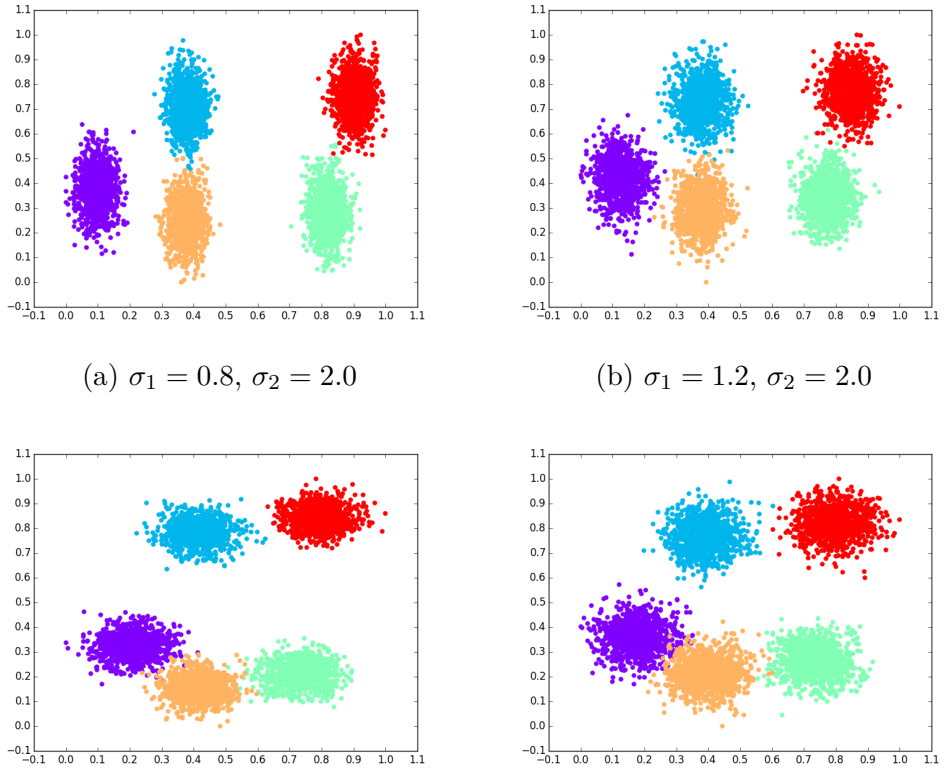
To measure the performance of the four potential solutions to address the shrink problem, we run experiments on a 2-gent network and K-means is used as clustering algorithm. Each agent initial their algorithm randomly ([0,0.3]) and the delay for sending a message is random as well ([0.5,1]). Data points inside each cluster are randomly generated from a binormal distribution (μ = [1, 5, 8, 13, 19, 3, 8, 2, 21, 14]) and they are scaled to the range [0,1] before assigning to network nodes randomly. Each nodes have 200 data points and the number of clusters in all simulations is set to 5. Figure 8.4 shows the four test datasets used in the experiment.

In addition to the shift distance of centroids, bounding boxes, rotations, we also measure the convergence time and accuracy against centralised method. The convergence time records the time interval from the first agent initial its work until the last agent finish its work. We use the percentage of membership mismatch (PMM) as our formal measure of accuracy.

$$PMM^{(i)} = 100\frac{|\vec{x} \in X^{(i)} : L_c(\vec{x}) \neq L_p^i(\vec{x})|}{|X^{(i)}|} \tag{8.1}$$

where $L_c(\vec{x})$ denotes the label of the cluster to which $\vec{x}$ is assigned at the end of centralized K-means and $L_p^i(\vec{x})$ denotes the label of the cluster to which $\vec{x}$ is assigned once the node reaches the termination state.

For different datasets, the shift distance, rotation and moving distance for three bounding boxes for each agent is measured. For accuracy measurement, min, max and median are also included in the brackets.



(a) $\sigma_1 = 0.8$, $\sigma_2 = 2.0$

(b) $\sigma_1 = 1.2$, $\sigma_2 = 2.0$

(c) $\sigma_1 = 2.0$, $\sigma_2 = 0.8$

(d) $\sigma_1 = 2.0$, $\sigma_2 = 1.2$

Figure 8.4: Dataset used in the experiment

Compare table 8.6 with table 8.7, table 8.8 and table 8.9, we can see that proposed algorithm without any make-up plan and with bounding line extension outperforms other schemes. Comparing to table 8.6, there is little difference in convergence time, but the moving distance of three bounding boxes and rotation increase, and accuracy decreases when we make up the shrink of outer box only (table 8.7). If we extend the inner boxes to midway to the next relevant point, the convergence time decreases. Although there is slightly increase in rotation (but with a higher standard deviation), shift distance of centroids and bounding boxes distance, the accuracy still remains high. When step 3 and 4 combined, the rotation increases with a higher standard deviation and accuracy decrease. The most possible reason lies in the re-generated data. Without any shrink make-up plan, data points of different clusters are tender to be more separated from each other, that is the reason why the clustering accuracy is high by this scheme.

Comparing table 8.6 with table 8.8, we can see that proposed algorithm without any plan to make up the shrink outperforms that with bounding line extension when clusters are well separated from each other (the first two rows). However,

Table 8.6: Default initial centroids, full K-means initialisation, No scheme to make up the shrink

Data	Measures	Mean	S.D
$\sigma_1 = 0.8$ $\sigma_2 = 2.0$	Centroids	[0.03, 0.012, 0.014, 0.019, 0.01] [0.015, 0.012, 0.029, 0.012, 0.017]	[0.09, 0.01, 0.02, 0.03, 0.01] [0.01, 0.01, 0.07, 0.01, 0.04]
	Rotation	[-0.939, -0.574, 0.596, -0.139, -0.955] [-1.235, -0.155, -1.901, -0.058, 1.093]	[9.33, 2.72, 3.49, 2.38, 3.24] [3.31, 2.38, 7.52, 2.35, 2.62]
	Bounding boxes	[[0.864, 0.832, 0.818], [1.109, 1.097, 1.094] 1.15, 1.138, 1.135], [1.412, 1.389, 1.379] [0.625, 0.609, 0.603]], [[0.861, 0.826, 0.808] [1.13, 1.115, 1.107], [1.199, 1.182, 1.169] [1.413, 1.377, 1.36], [0.668, 0.646, 0.632]]	[[0.39, 0.39, 0.38], [0.4, 0.38, 0.38] [0.34, 0.33, 0.31], [0.38, 0.36, 0.35] [0.29, 0.27, 0.27]], [[0.42, 0.4, 0.4] [0.4, 0.39, 0.38], [0.27, 0.27, 0.27] [0.41, 0.39, 0.39], [0.27, 0.27, 0.26]]
	Time	2.18	0.164
	Accuracy	98.8 [87.0 100.0 100.0]	3.61
$\sigma_1 = 1.2$ $\sigma_2 = 2.0$	Centroids	[0.019, 0.015, 0.009, 0.01, 0.029] [0.01, 0.008, 0.009, 0.013, 0.011]	[0.03, 0.02, 0.01, 0.01, 0.09] [0.01, 0.0, 0.0, 0.01, 0.01]
	Rotation	[2.146, -1.162, 0.45, -0.34, -1.329] [1.501, 0.301, 0.007, -1.045, 0.511]	[9.17, 5.51, 4.19, 3.23, 7.94] [3.08, 2.79, 2.89, 3.74, 4.63]
	Bounding boxes	[[1.123, 1.115, 1.107], [1.101, 1.089, 1.085] [1.198, 1.185, 1.179], [1.107, 1.1, 1.094] [0.625, 0.62, 0.617]], [[1.033, 1.033, 1.031] [1.097, 1.089, 1.085], [1.194, 1.184, 1.178] [1.118, 1.108, 1.097], [0.618, 0.612, 0.611]]	[[0.45, 0.44, 0.44], [0.44, 0.45, 0.45] [0.34, 0.35, 0.36], [0.26, 0.26, 0.26] [0.23, 0.22, 0.21]], [[0.44, 0.44, 0.44] [0.45, 0.45, 0.45], [0.38, 0.38, 0.38] [0.27, 0.26, 0.26], [0.22, 0.21, 0.21]]
	Time	2.19	0.16
	Accuracy	99.33 [86.5 100.0 100.0]	2.942
$\sigma_1 = 2.0$ $\sigma_2 = 0.8$	Centroids	[0.019, 0.022, 0.017, 0.037, 0.036] [0.016, 0.014, 0.01, 0.036, 0.011]	[0.03, 0.03, 0.03, 0.11, 0.11] [0.03, 0.02, 0.01, 0.11, 0.01]
	Rotation	[1.831, -1.996, -5.232, -1.817, -4.576] [2.545, 2.182, 1.963, -1.326, -0.09]	[12.71, 12.84, 16.18, 16.83, 11.55] [15.67, 11.0, 8.64, 17.24, 13.02]
	Bounding boxes	[[0.563, 0.555, 0.55], [0.478, 0.468, 0.459] [0.607, 0.583, 0.572], [0.551, 0.526, 0.514] [0.412, 0.402, 0.392]], [[0.584, 0.574, 0.567] [0.47, 0.466, 0.461], [0.582, 0.571, 0.564] [0.463, 0.452, 0.449], [0.342, 0.342, 0.33]]	[[0.25, 0.24, 0.24], [0.24, 0.23, 0.23] [0.31, 0.31, 0.31], [0.38, 0.37, 0.37] [0.47, 0.47, 0.47]], [[0.27, 0.27, 0.27] [0.34, 0.33, 0.33], [0.33, 0.33, 0.32] [0.37, 0.37, 0.37], [0.44, 0.44, 0.44]]
	Time	2.17	0.173
	Accuracy	98.25 [78.0 100.0 100.0]	5.439
$\sigma_1 = 2.0$ $\sigma_2 = 1.2$	Centroids	[0.059, 0.027, 0.013, 0.051, 0.014] [0.065, 0.015, 0.017, 0.029, 0.024]	[0.15, 0.04, 0.01, 0.13, 0.01] [0.16, 0.02, 0.02, 0.07, 0.05]
	Rotation	[8.137, -7.688, 8.652, 8.791, 14.181] [-2.949, -8.15, 7.109, 2.533, 8.885]	[39.8, 42.66, 32.01, 34.51, 26.78] [32.28, 32.39, 43.19, 22.61, 28.47]
	Bounding boxes	[[0.46, 0.437, 0.416], [0.551, 0.517, 0.494] [0.533, 0.515, 0.5], [0.566, 0.558, 0.554] [0.438, 0.419, 0.4]], [[0.503, 0.472, 0.449] [0.395, 0.373, 0.357], [0.553, 0.535, 0.506] [0.595, 0.577, 0.565], [0.547, 0.524, 0.503]]	[[0.31, 0.3, 0.29], [0.3, 0.3, 0.29] [0.19, 0.18, 0.18], [0.25, 0.24, 0.23] [0.2, 0.19, 0.19]], [[0.27, 0.26, 0.25] [0.25, 0.23, 0.22], [0.24, 0.23, 0.21] [0.14, 0.14, 0.15], [0.34, 0.33, 0.32]]
	Time	2.19	0.208
	Accuracy	97.38 [73.0 100.0 100.0]	6.769

when there are overlaps among clusters, the later scheme outperforms the previous scheme (the last two rows). In some practical applications, there are slightly overlap, even huge overlap between clusters, we used the scheme that extending the bounding line to half way to the closet points to address the shrink up problem.

8.1.6 Summary

We proposed representing a cluster by its centroids, percentiles and PCs. Simply use centroids to represent a cluster is not enough, because we only know the central place but have no idea about how data points spread inside this cluster.

Table 8.7: Default initial centroids, full K-means initialisation, make up the shrink of 100% bounding box only

Data	Measures	Mean	S.D
$\sigma_1 = 0.8$ $\sigma_2 = 2.0$	Centroids	[0.185, 0.038, 0.031, 0.087, 0.233] [0.267, 0.163, 0.064, 0.046, 0.278]	[0.31, 0.05, 0.04, 0.19, 0.31] [0.38, 0.33, 0.11, 0.08, 0.38]
	Rotation	[-3.909, -1.292, -4.503, 5.6, -0.415] [3.883, -0.267, -5.787, -2.398, -12.016]]	[20.76, 10.83, 13.84, 13.33, 21.07] [14.62, 9.64, 9.06, 8.43, 14.63]
	Bounding boxes	[[1.587, 1.405, 1.256], [1.237, 1.17, 1.109] [1.348, 1.242, 1.226], [1.614, 1.509, 1.456] [1.433, 1.181, 0.988]], [[1.516, 1.316, 1.182] [1.319, 1.26, 1.175], [1.509, 1.475, 1.395] [1.627, 1.547, 1.552], [1.242, 1.014, 0.907]]	[[0.56, 0.58, 0.62], [0.37, 0.38, 0.39] [0.52, 0.46, 0.4], [0.28, 0.3, 0.37] [0.45, 0.44, 0.45]], [[0.44, 0.5, 0.54] [0.37, 0.36, 0.38], [0.57, 0.53, 0.56] [0.55, 0.61, 0.58], [0.54, 0.56, 0.54]]
	Time	2.12	0.143
	Accuracy	91.42 [77.5 100.0 91.0]	7.42
$\sigma_1 = 1.2$ $\sigma_2 = 2.0$	Centroids	[0.07, 0.111, 0.019, 0.074, 0.125] [0.131, 0.207, 0.069, 0.068, 0.2]	[0.14, 0.26, 0.02, 0.19, 0.21] [0.27, 0.35, 0.21, 0.08, 0.34]
	Rotation	[-0.104, -0.771, 4.863, -0.95, 8.911] [-0.013, 4.812, -0.523, 2.542, 2.9]	[10.11, 8.41, 9.43, 11.3, 20.78] [11.01, 13.58, 11.39, 12.92, 20.48]
	Bounding boxes	[[1.441, 1.282, 1.159], [1.339, 1.269, 1.218] [1.335, 1.324, 1.271], [1.208, 1.106, 1.065] [1.24, 1.011, 0.865]], [[1.368, 1.274, 1.19] [1.473, 1.327, 1.245], [1.328, 1.285, 1.271] [1.404, 1.311, 1.284], [1.167, 1.027, 0.951]]	[[0.3, 0.36, 0.5], [0.41, 0.45, 0.5] [0.28, 0.26, 0.31], [0.21, 0.24, 0.28] [0.41, 0.36, 0.41]], [[0.37, 0.42, 0.48] [0.3, 0.44, 0.54], [0.28, 0.33, 0.32] [0.58, 0.62, 0.66], [0.48, 0.51, 0.53]]
	Time	2.13	0.203
	Accuracy	92.13 [79.0 100.0 90.75]	5.94
$\sigma_1 = 2.0$ $\sigma_2 = 0.8$	Centroids	[0.102, 0.114, 0.315, 0.071, 0.348] [0.141, 0.123, 0.098, 0.118, 0.508]	[0.29, 0.34, 0.48, 0.23, 0.47] [0.37, 0.27, 0.24, 0.32, 0.55]
	Rotation	[-4.338, -12.95, -8.427, -9.997, -14.016] [-4.781, -12.943, -9.903, -1.845, -17.031]	[20.62, 21.62, 23.41, 23.59, 27.37] [19.92, 27.3, 25.67, 21.24, 23.13]
	Bounding boxes	[[1.041, 0.879, 0.767], [0.858, 0.698, 0.622] [1.199, 1.054, 1.017], [0.755, 0.711, 0.667] [1.427, 1.314, 1.21]], [[0.903, 0.811, 0.739] [1.137, 0.792, 0.692], [0.948, 0.822, 0.709] [0.873, 0.818, 0.767], [1.711, 1.595, 1.566]]	[[0.56, 0.52, 0.5], [0.77, 0.66, 0.66] [0.81, 0.79, 0.8], [0.54, 0.54, 0.53] [0.81, 0.87, 0.92]], [[0.5, 0.48, 0.47] [0.63, 0.55, 0.55], [0.58, 0.49, 0.51] [0.69, 0.7, 0.68], [0.85, 0.94, 0.96]]
	Time	2.07	0.131
	Accuracy	88.35 [81.5 93.5 90.5]	4.591
$\sigma_1 = 2.0$ $\sigma_2 = 1.2$	Centroids	[0.165, 0.34, 0.175, 0.137, 0.267] [0.057, 0.058, 0.107, 0.167, 0.224]]	[0.35, 0.52, 0.32, 0.28, 0.37] [0.06, 0.08, 0.21, 0.34, 0.37]
	Rotation	[-3.511, -7.871, 13.386, 20.445, 12.057] [-1.176, 0.506, 13.576, 6.833, 16.004]	[37.23, 35.27, 39.67, 41.93, 36.7] [39.22, 40.32, 39.59, 32.54, 34.44]
	Bounding boxes	[[1.135, 1.001, 0.89], [1.318, 1.181, 1.06] [0.979, 0.751, 0.635], [1.073, 0.894, 0.796] [1.511, 1.25, 1.132]], [[1.17, 0.921, 0.734] [1.028, 0.874, 0.772], [0.931, 0.782, 0.644] [0.805, 0.698, 0.612], [1.303, 1.086, 0.877]]	[[0.92, 0.93, 0.93], [0.91, 0.94, 0.97] [0.39, 0.21, 0.13], [0.73, 0.71, 0.73] [0.88, 0.97, 1.03]], [[0.56, 0.57, 0.5] [0.74, 0.75, 0.76], [0.35, 0.22, 0.16] [0.3, 0.22, 0.19], [0.73, 0.71, 0.74]]
	Time	2.12	0.13
	Accuracy	88.6 [76.5 100.0 90.0]	8.64

Percentiles and PCs are introduced to describe where data points are located and how data points spread. Experiments show that, compared to the original cluster, the new cluster we generated is similar. Descriptions for clusters that are elliptical are more stable than those for circular clusters. The rotation degree required to remap the new PCs from old PCs is heavily relying on the boundary points. One possible approach to improve the quality of generated cluster is choosing an appropriate percentile.

To conclude, we could generate a similar cluster based on its summary description generated by our method. However, there is a shrink problem about the bounding boxes of regenerated data points. To address this problem, we proposed that

Table 8.8: Default initial centroids, full K-means initialisation, extend the bounding line to half way to its closet point

Data	Measures	Mean	S.D
$\sigma_1 = 0.8$ $\sigma_2 = 2.0$	Centroids	[0.041, 0.061, 0.045, 0.039, 0.053], [0.008, 0.01, 0.015, 0.011, 0.012]	[0.1, 0.13, 0.1, 0.1, 0.11], [0.01, 0.01, 0.01, 0.01, 0.01]
	Rotation	[-1.785, -2.08, -1.155, -1.22, -2.88], [-0.768, 0.416, 0.438, -0.706, -0.484]	[7.97, 10.13, 12.24, 5.18, 7.37], [3.22, 3.19, 1.71, 2.96, 2.06]
	Bounding boxes	[[1.132, 0.171, 0.139], [0.891, 0.209, 0.172], [1.287, 0.191, 0.166], [1.324, 0.316, 0.282], [0.902, 0.31, 0.283]], [[1.109, 0.098, 0.078], [0.891, 0.18, 0.14], [1.29, 0.178, 0.141], [1.202, 0.182, 0.145], [0.758, 0.147, 0.121]]	[[0.39, 0.21, 0.18], [0.35, 0.25, 0.21], [0.42, 0.24, 0.21], [0.43, 0.51, 0.5], [0.53, 0.48, 0.48]], [[0.38, 0.16, 0.14], [0.35, 0.24, 0.19], [0.44, 0.23, 0.19], [0.39, 0.22, 0.2], [0.39, 0.2, 0.18]]
	Time	2.05	0.13
	Accuracy	97.6 [87.5 100.0 100.0]	4.68
$\sigma_1 = 1.2$ $\sigma_2 = 2.0$	Centroids	[0.037, 0.037, 0.018, 0.044, 0.05] [0.013, 0.032, 0.012, 0.033, 0.016]	[0.09, 0.09, 0.02, 0.09, 0.11] [0.02, 0.09, 0.01, 0.08, 0.03]
	Rotation	[-0.011, -3.917, -4.047, -2.21, -1.209] [-0.138, -3.271, -0.525, -0.532, -3.61]	[7.14, 11.67, 12.49, 11.35, 10.73] [5.74, 7.57, 2.54, 4.61, 8.99]
	Bounding boxes	[[1.179, 0.159, 0.119], [1.001, 0.211, 0.176], [1.185, 0.241, 0.199], [1.302, 0.455, 0.388], [0.821, 0.283, 0.22]], [[1.158, 0.125, 0.089], [1.045, 0.276, 0.239], [1.151, 0.132, 0.102], [1.133, 0.266, 0.192], [0.771, 0.195, 0.153]]	[[0.44, 0.22, 0.17], [0.45, 0.27, 0.22], [0.41, 0.27, 0.22], [0.35, 0.49, 0.49], [0.28, 0.28, 0.22]], [[0.46, 0.19, 0.14], [0.42, 0.34, 0.31], [0.38, 0.18, 0.16], [0.37, 0.3, 0.23], [0.29, 0.23, 0.2]]
	Time	1.99	0.129
	Accuracy	97.3 [84.5 100.0 100.0]	5.96
$\sigma_1 = 2.0$ $\sigma_2 = 0.8$	Centroids	[0.036, 0.014, 0.008, 0.01, 0.018], [0.008, 0.009, 0.009, 0.009, 0.008]	[0.11, 0.02, 0.01, 0.01, 0.04] [0.0, 0.0, 0.0, 0.01, 0.01]
	Rotation	[-6.068, 3.23, 1.566, 0.117, 0.649] [-4.632, 7.785, -0.19, -1.469, 1.916]	[23.63, 15.39, 23.6, 7.12, 7.51] [24.0, 19.15, 24.39, 9.97, 13.7]
	Bounding boxes	[[0.414, 0.209, 0.177], [0.473, 0.185, 0.145] [0.437, 0.189, 0.158], [0.487, 0.256, 0.201] [0.364, 0.188, 0.158]], [[0.378, 0.15, 0.126] [0.464, 0.165, 0.131], [0.426, 0.185, 0.153] [0.448, 0.213, 0.168], [0.323, 0.163, 0.129]]	[[0.28, 0.17, 0.15], [0.26, 0.19, 0.15] [0.31, 0.17, 0.15], [0.25, 0.23, 0.18] [0.35, 0.34, 0.32]], [[0.31, 0.16, 0.14] [0.28, 0.15, 0.13], [0.28, 0.17, 0.15] [0.24, 0.22, 0.18], [0.24, 0.17, 0.14]]
	Time	2.08	0.26
	Accuracy	99.42 [88.5 100.0 100.0]	2.506
$\sigma_1 = 2.0$ $\sigma_2 = 1.2$	Centroids	[0.01, 0.014, 0.036, 0.014, 0.018] [0.008, 0.022, 0.012, 0.013, 0.034]	[0.0, 0.01, 0.1, 0.01, 0.04] [0.01, 0.04, 0.01, 0.01, 0.1]
	Rotation	[5.703, 0.541, -8.263, 14.825, -10.378] [-0.533, -8.295, -15.307, 11.843, -4.959]	[26.56, 14.9, 26.18, 26.99, 29.97] [23.55, 19.28, 22.95, 33.49, 27.54]
	Bounding boxes	[[0.836, 0.248, 0.202], [0.768, 0.284, 0.239] [0.792, 0.211, 0.184], [0.626, 0.217, 0.174] [0.581, 0.278, 0.238]], [[0.777, 0.233, 0.197] [0.749, 0.301, 0.266], [0.774, 0.229, 0.19] [0.616, 0.235, 0.192], [0.586, 0.307, 0.258]]	[[0.38, 0.2, 0.17], [0.32, 0.19, 0.16] [0.51, 0.26, 0.24], [0.37, 0.18, 0.16] [0.22, 0.22, 0.2]], [[0.46, 0.21, 0.18] [0.33, 0.3, 0.29], [0.53, 0.2, 0.16] [0.41, 0.19, 0.17], [0.19, 0.21, 0.18]]
	Time	2.08	0.176
	Accuracy	98.42 [86.0 100.0 100.0]	4.157

extend bounding line of inside boxes to half way to the closet points. One effect way to generate a similar cluster is to choose the appropriate percentile.

Table 8.9: Default initial centroids, full K-means initialisation, combination of scheme 2 and 3

Data	Measures	Mean	S.D
$\sigma_1 = 0.8$ $\sigma_2 = 2.0$	Centroids	[0.159, 0.078, 0.034, 0.075, 0.231] [0.121, 0.152, 0.109, 0.11, 0.29]	[0.25, 0.2, 0.03, 0.23, 0.39] [0.23, 0.27, 0.27, 0.27, 0.39]
	Rotation	[3.78, -5.591, -1.771, 1.85, -5.899] [-2.1, -5.029, -1.147, 0.292, -5.574]	[22.96, 13.05, 13.21, 7.69, 14.16] [21.37, 9.06, 15.28, 11.62, 21.97]
	Bounding boxes	[[1.592, 0.897, 0.731], [1.408, 0.49, 0.298], [1.172, 0.505, 0.319], [1.461, 0.438, 0.291], [1.461, 1.067, 0.937]], [[1.379, 0.807, 0.557], [1.544, 0.737, 0.62], [1.282, 0.445, 0.328], [1.559, 0.664, 0.563], [1.477, 1.054, 0.879]]	[[0.45, 0.72, 0.74], [0.38, 0.47, 0.28], [0.44, 0.46, 0.32], [0.31, 0.36, 0.25], [0.65, 0.91, 0.97]], [[0.43, 0.63, 0.55], [0.47, 0.72, 0.73], [0.41, 0.5, 0.49] [0.45, 0.82, 0.82], [0.56, 0.79, 0.84]]
	Time	2.05	0.121
	Accuracy	90.42 [81.0 100.0 91.0]	5.88
$\sigma_1 = 1.2$ $\sigma_2 = 2.0$	Centroids	[0.177, 0.146, 0.082, 0.065, 0.278] [0.044, 0.136, 0.069, 0.046, 0.216]	[0.3, 0.27, 0.24, 0.17, 0.34] [0.04, 0.24, 0.2, 0.09, 0.31]
	Rotation	[-6.054, 4.113, 1.265, -4.891, -4.489] [0.344, 2.099, 3.232, -0.412, 1.755]	[18.75, 16.4, 15.89, 18.13, 21.94] [16.91, 17.98, 11.76, 12.23, 19.19]
	Bounding boxes	[[1.298, 0.67, 0.568], [1.628, 0.908, 0.806] [1.259, 0.338, 0.219], [1.119, 0.376, 0.248] [1.504, 1.23, 1.062]], [[1.325, 0.68, 0.524] [1.398, 0.542, 0.405], [1.285, 0.414, 0.255] [1.231, 0.512, 0.364], [1.437, 1.083, 0.841]]	[[0.6, 0.81, 0.81], [0.57, 0.88, 0.89] [0.35, 0.26, 0.2], [0.35, 0.35, 0.22] [0.57, 0.8, 0.87]], [[0.39, 0.57, 0.53] [0.28, 0.54, 0.48], [0.34, 0.29, 0.2] [0.45, 0.59, 0.56], [0.45, 0.66, 0.71]]
	Time	1.99	0.11
	Accuracy	91.8 [77.5 100.0 91.5]	6.67
$\sigma_1 = 2.0$ $\sigma_2 = 0.8$	Centroids	[0.249, 0.109, 0.172, 0.101, 0.52] [0.202, 0.058, 0.126, 0.02, 0.694]	[0.43, 0.28, 0.39, 0.3, 0.49] [0.36, 0.07, 0.27, 0.03, 0.57]
	Rotation	[-5.391, 0.829, -0.044, -3.497, -10.434] [-1.942, -4.779, 1.456, -1.148, -4.535]	[26.96, 23.58, 24.61, 24.06, 28.77] [35.12, 27.12, 32.19, 23.34, 35.62]
	Bounding boxes	[[1.243, 0.969, 0.911], [1.106, 0.794, 0.691] [1.025, 0.795, 0.743], [0.645, 0.374, 0.336] [1.061, 0.927, 0.805]], [[0.968, 0.613, 0.535] [0.842, 0.632, 0.529], [0.867, 0.456, 0.382] [0.69, 0.431, 0.329], [1.787, 1.635, 1.541]]	[[0.86, 1.05, 1.06], [1.01, 1.04, 1.05] [0.87, 1.01, 1.03], [0.58, 0.57, 0.56] [0.73, 0.77, 0.81]], [[0.58, 0.67, 0.66] [0.71, 0.73, 0.68], [0.59, 0.5, 0.48] [0.51, 0.56, 0.55], [0.92, 1.08, 1.14]]
	Time	2.06	0.252
	Accuracy	86.35 [78.5 100.0 84.25]	5.229
$\sigma_1 = 2.0$ $\sigma_2 = 1.2$	Centroids	[0.214, 0.293, 0.019, 0.232, 0.317], [0.127, 0.171, 0.017, 0.321, 0.36]	[0.38, 0.47, 0.01, 0.37, 0.48] [0.28, 0.24, 0.02, 0.45, 0.49]
	Rotation	[-0.437, -1.631, 2.609, -3.575, -7.883] [-14.968, -7.464, -7.468, 3.374, 8.479]	[27.69, 29.45, 28.92, 33.01, 29.5] [25.76, 27.63, 24.11, 29.91, 35.68]
	Bounding boxes	[[1.245, 0.623, 0.538], [1.508, 1.231, 1.157] [0.973, 0.426, 0.319], [1.265, 0.805, 0.692] [1.28, 1.038, 0.922]], [[1.179, 0.542, 0.496] [1.335, 0.971, 0.907], [1.027, 0.398, 0.277] [1.344, 0.901, 0.763], [1.386, 1.13, 0.976]]	[[0.66, 0.71, 0.73], [0.93, 1.06, 1.09] [0.42, 0.24, 0.16], [0.99, 1.07, 1.09] [0.85, 0.96, 0.99]], [[0.56, 0.69, 0.71] [0.82, 1.0, 1.01], [0.47, 0.38, 0.21] [0.6, 0.74, 0.74], [0.7, 0.84, 0.89]]
	Time	2.01	0.137
	Accuracy	86.33 [71.0 100.0 84.0]	7.025

8.2 Comparison result on single pattern sparse networks

8.2.1 K-means as clustering algorithm

Figure 8.5, 8.6 and 8.7 below show the results on data generated from fully symmetric Gaussians, partially-symmetric Gaussians and rectangular uniform distributions when the underlying network is sparse. The density of the network mainly affects the number of messages received. Comparing figure 8.5 with figure 4.6, all

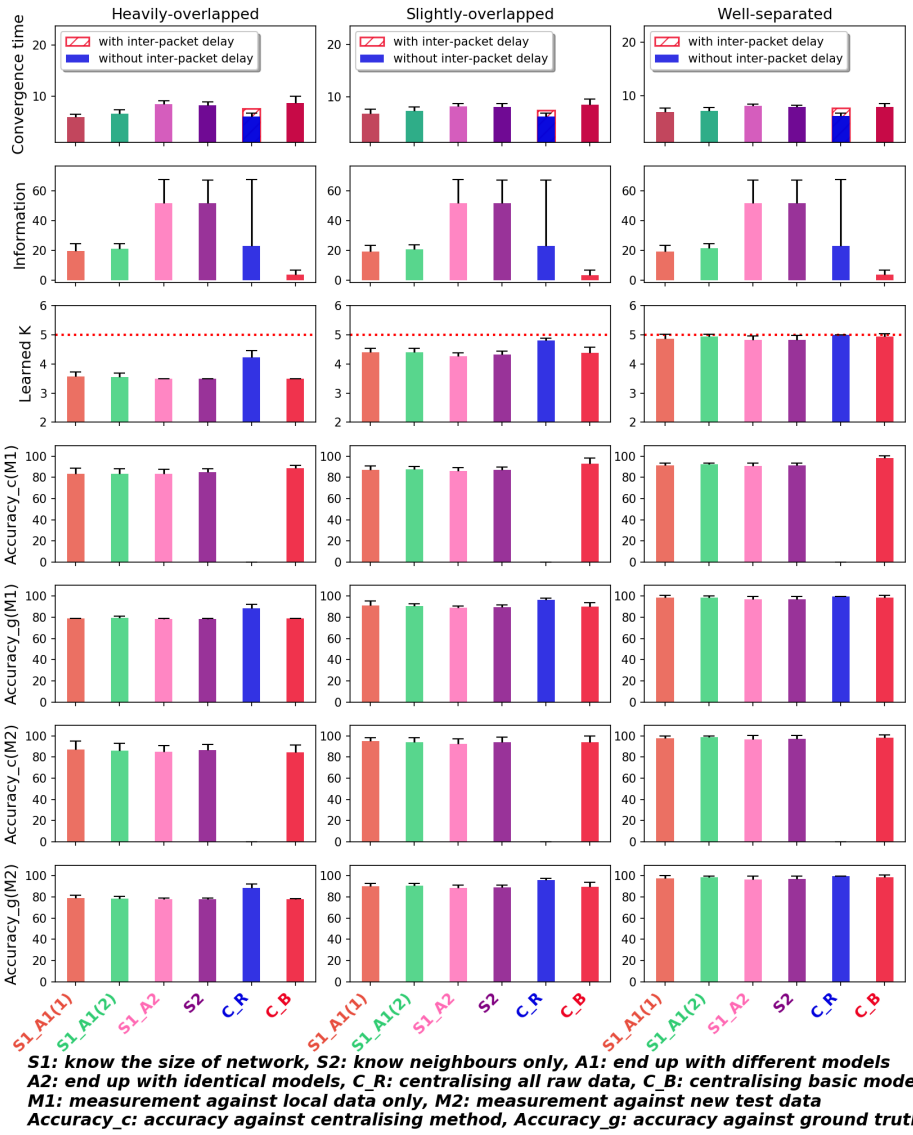


Figure 8.5: K-means with fully symmetric Gaussians on sparse network

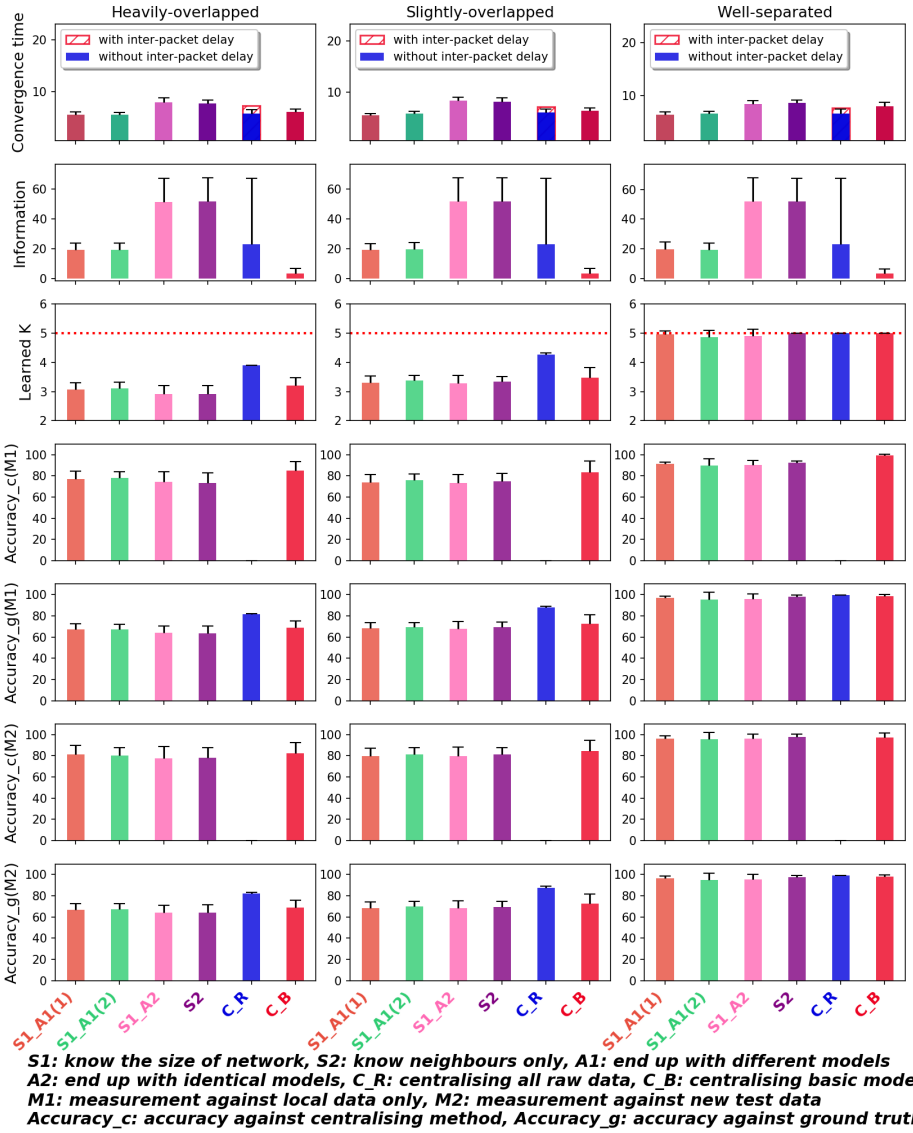


Figure 8.6: K-means on sparse network with partially symmetric Gaussians

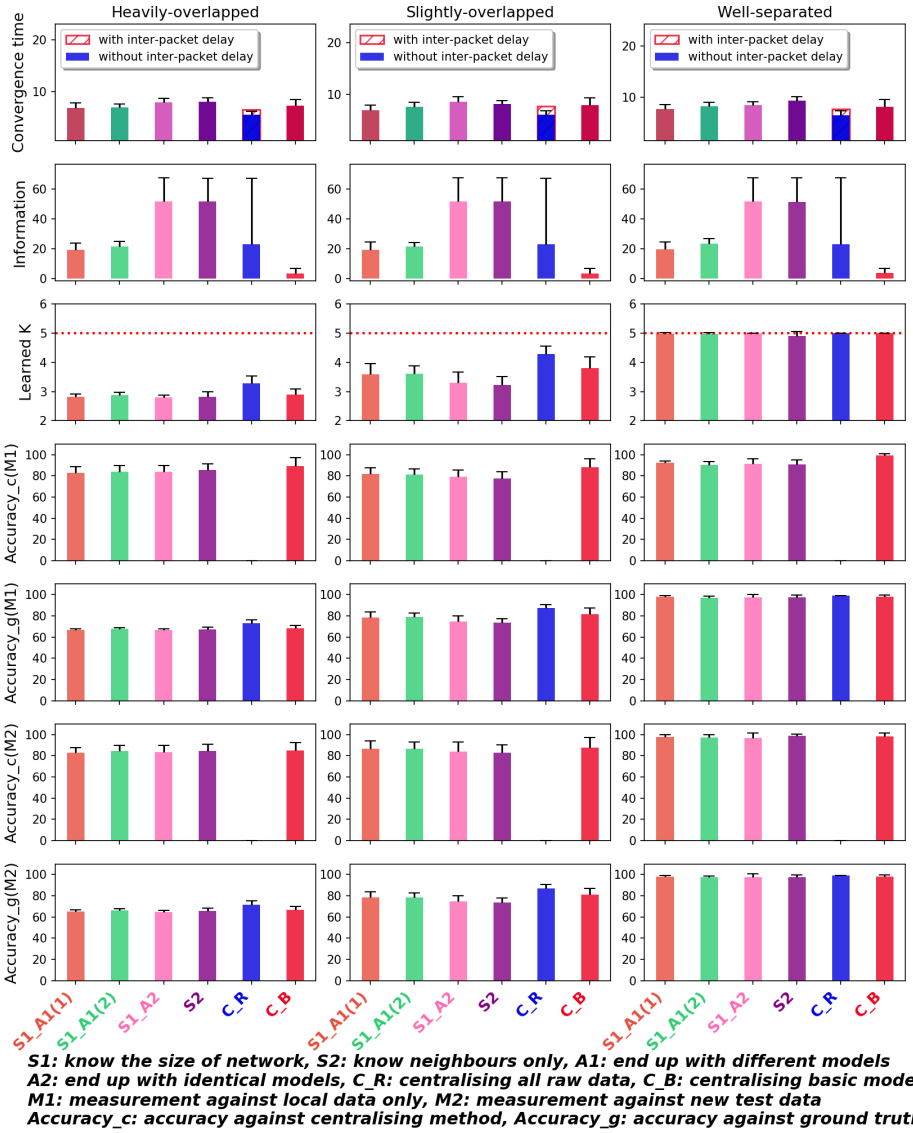
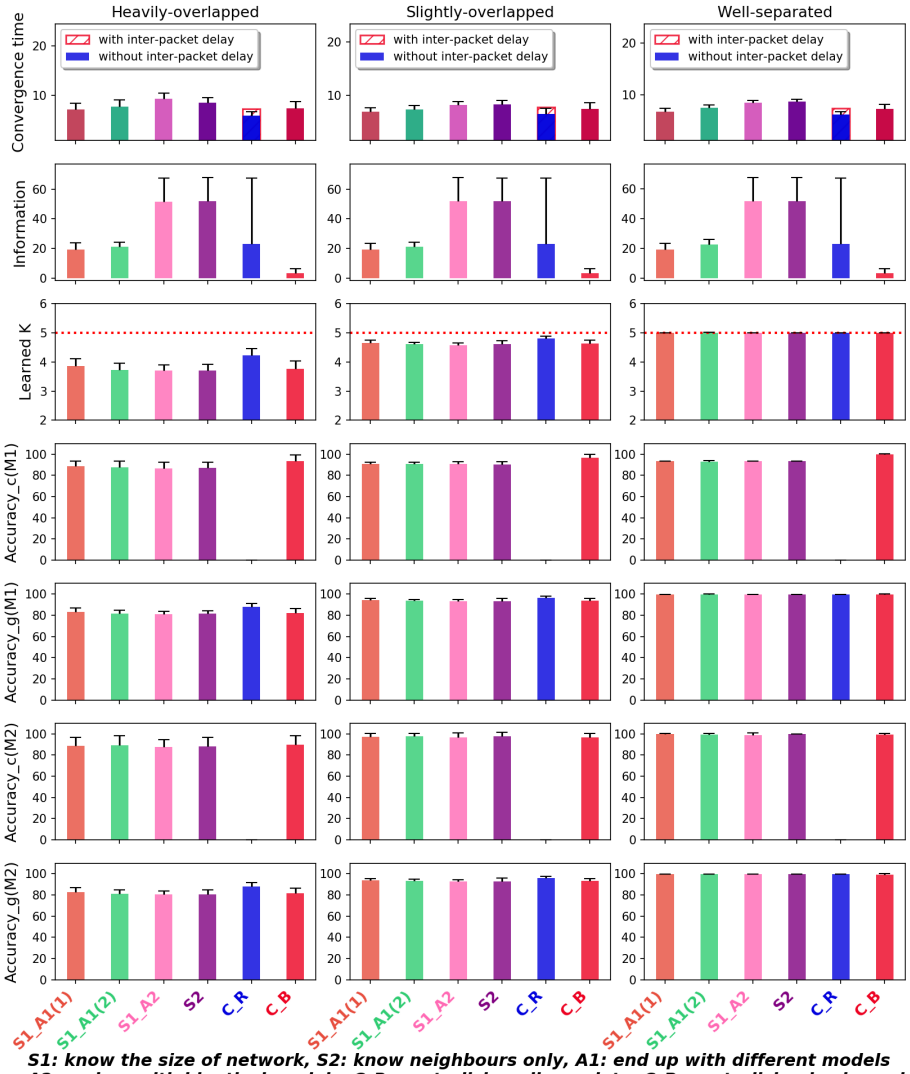


Figure 8.7: K-means on sparse network with uniform distributed dataset



S1: know the size of network, S2: know neighbours only, A1: end up with different models
A2: end up with identical models, C_R: centralising all raw data, C_B: centralising basic models
M1: measurement against local data only, M2: measurement against new test data
Accuracy_c: accuracy against centralising method, Accuracy_g: accuracy against ground truth

Figure 8.8: SG on sparse network with fully symmetric Gaussians

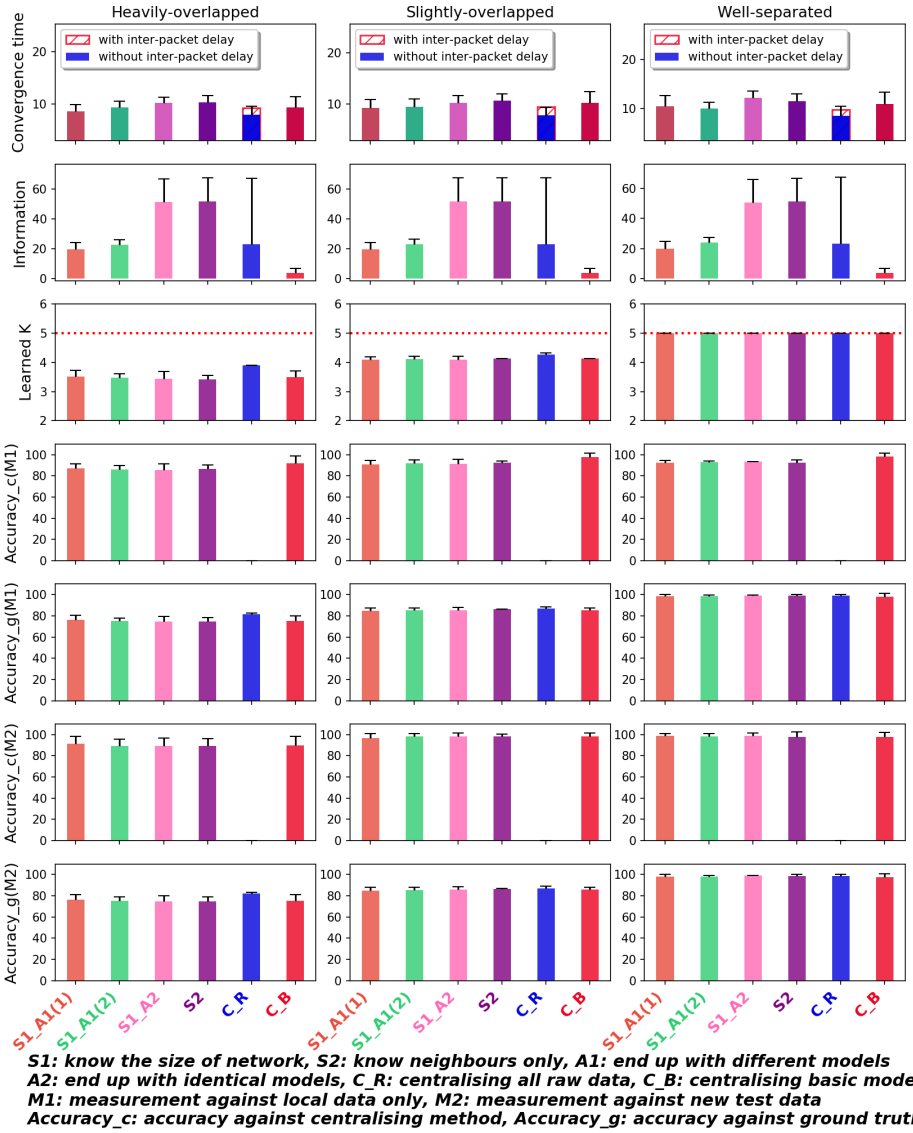


Figure 8.9: SG on sparse network with partially symmetric Gaussians

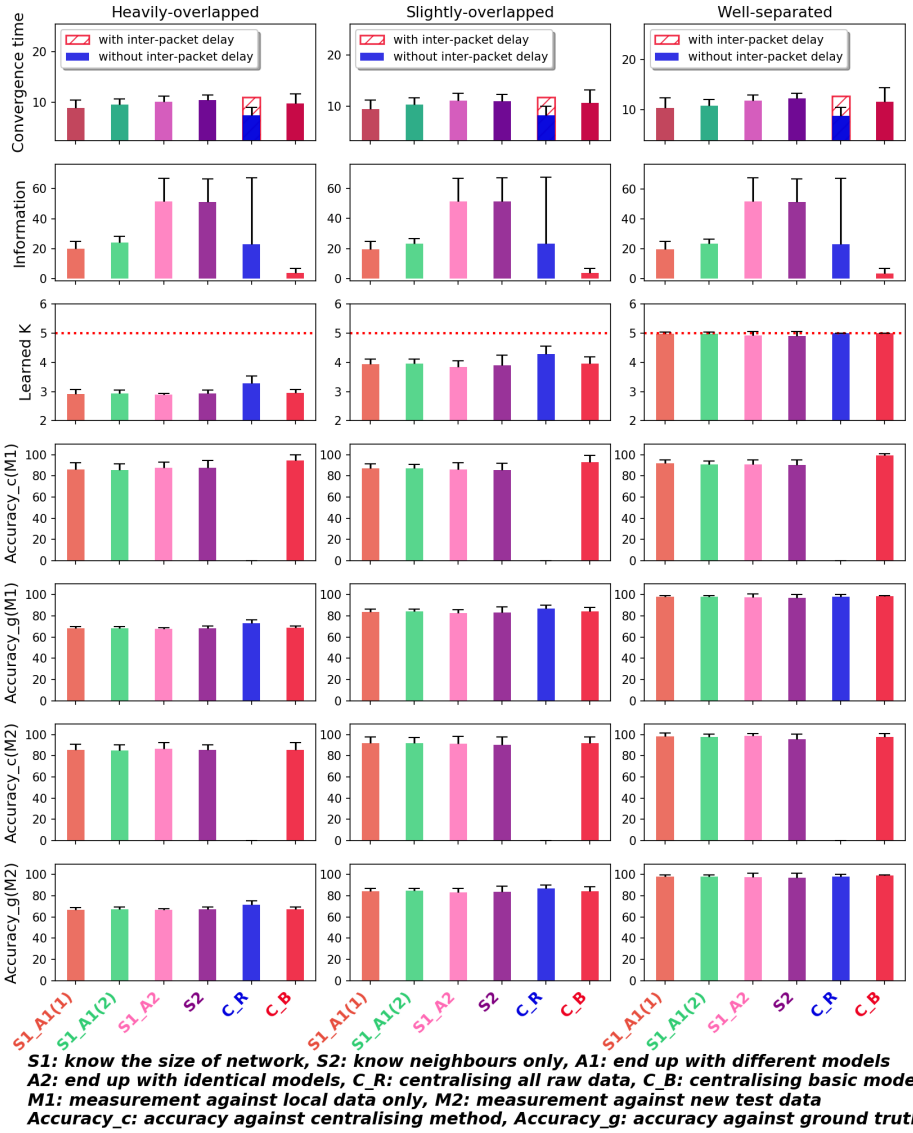


Figure 8.10: SG on sparse network with uniform distributed dataset

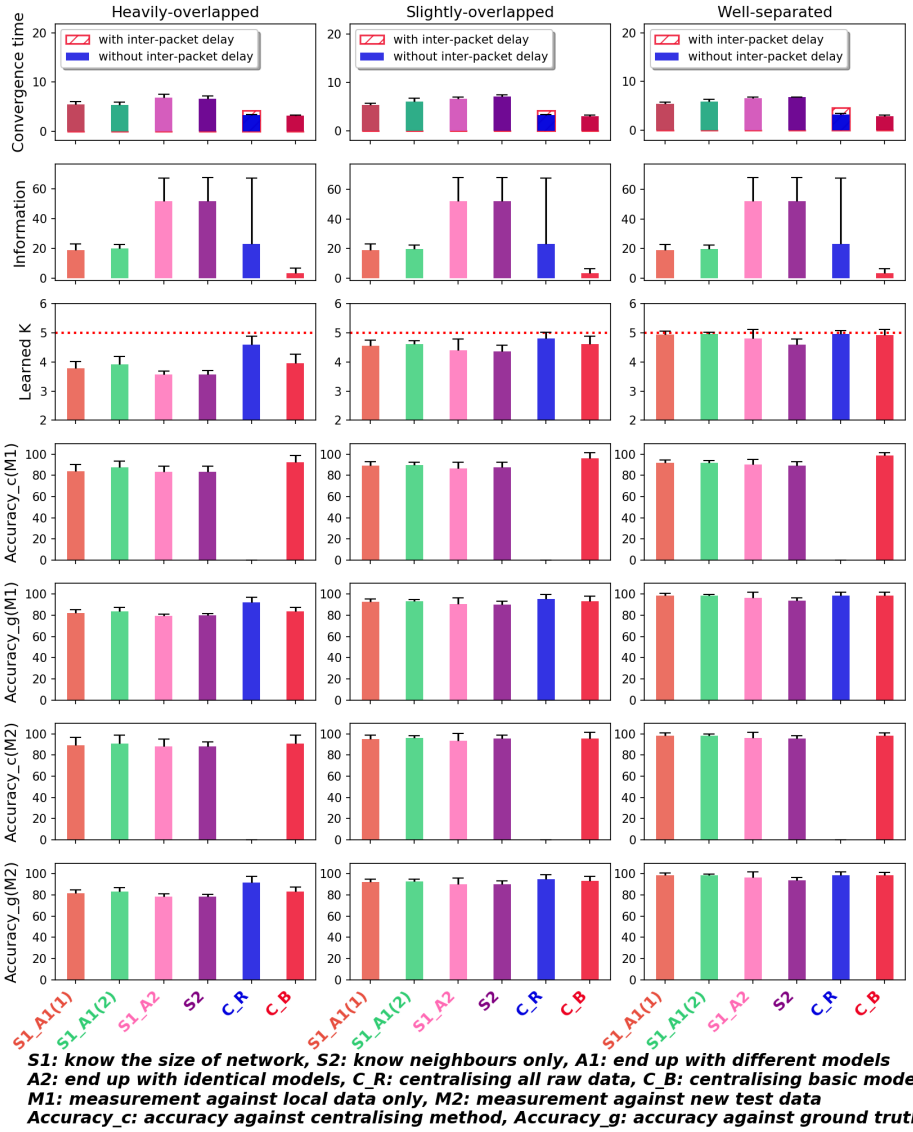


Figure 8.11: GMM on sparse network with fully symmetric Gaussians

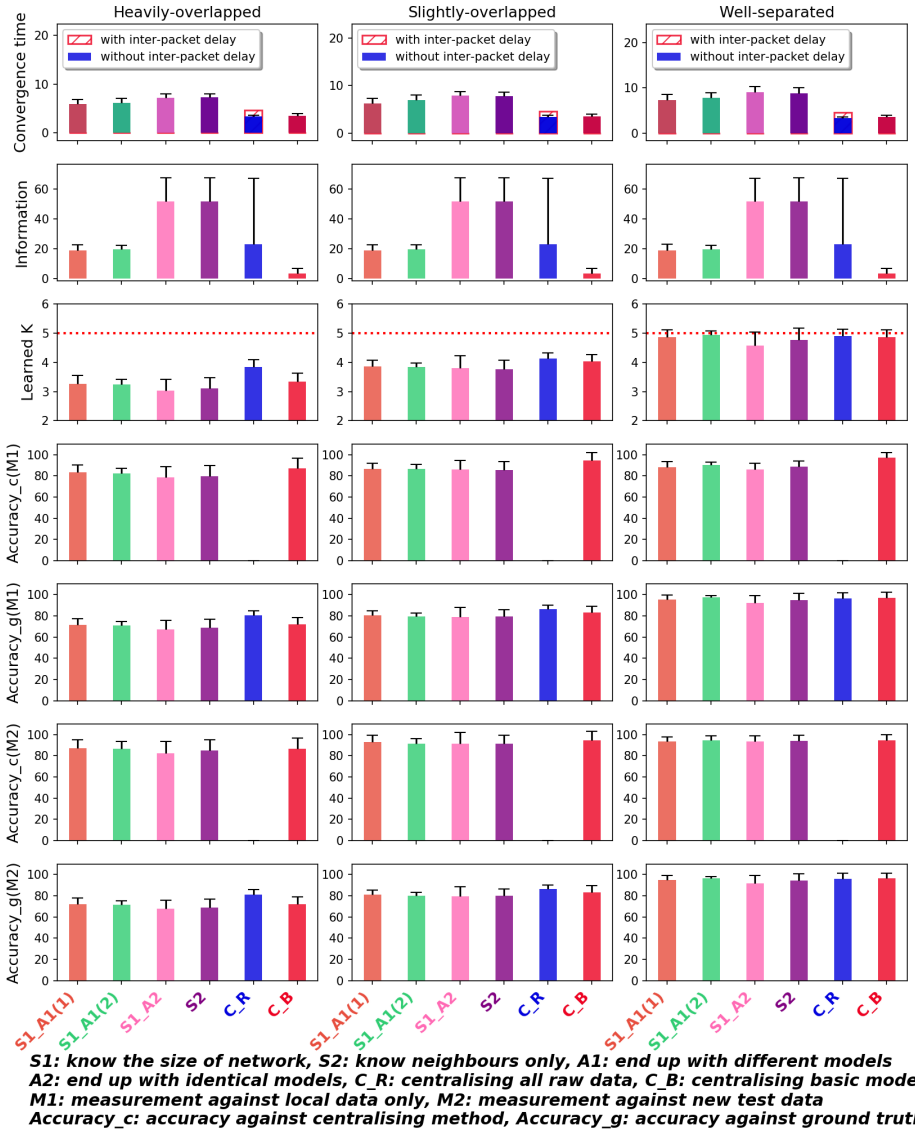


Figure 8.12: GMM on sparse network with partially symmetric Gaussians

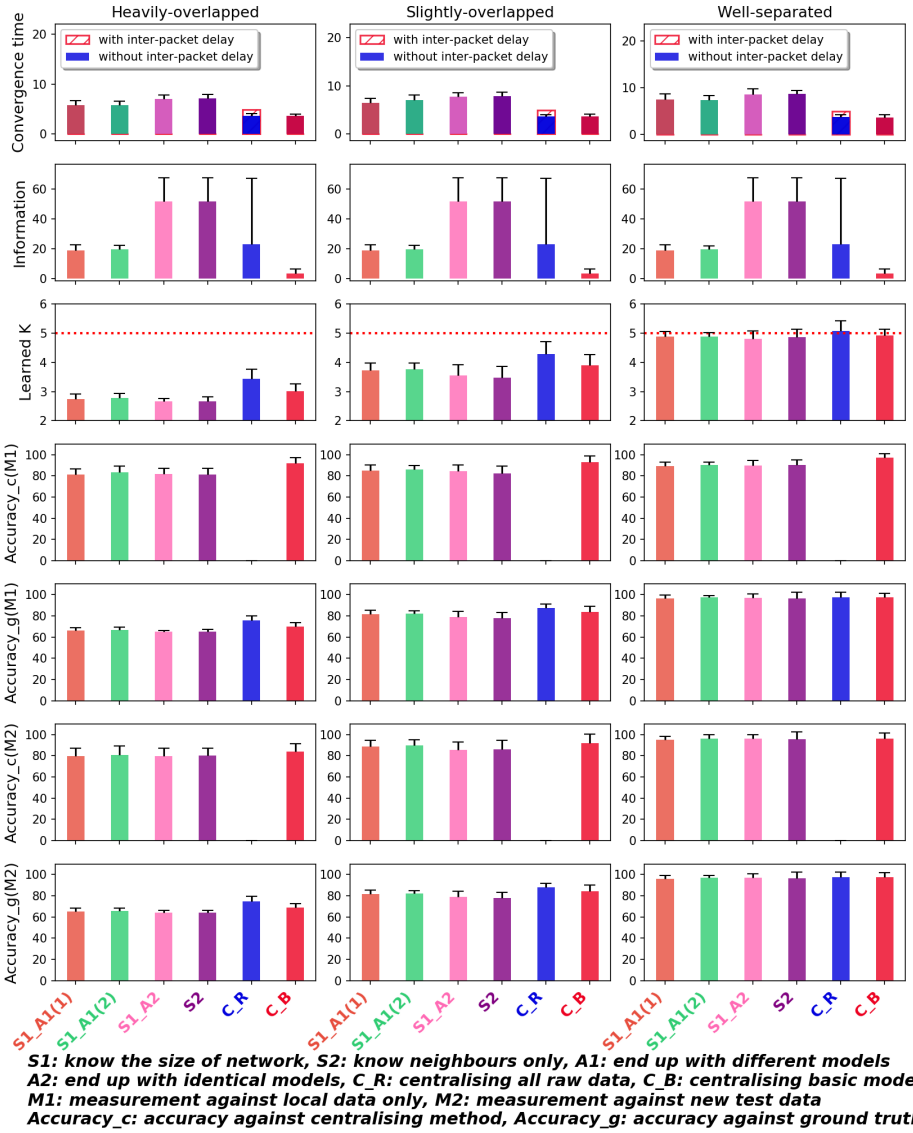


Figure 8.13: GMM on sparse network with uniform distributed dataset

8.3 Comparison result on similarity measurements

In this section, we show the comparison result on similarity measurements when GMM and SG are used to cluster data points that generated from fully symmetric Gaussians and uniform distributions. The results on partially symmetric Gaussian are omitted since they show the same behaviour as that with fully symmetric Gaussians.

8.3.1 GMM as clustering algorithm

First, we show the performance when GMM is used as the clustering algorithm. Figure 8.14 shows the comparison result of accuracy against centralised GMM when data points are generated from fully symmetric Gaussians. There are little difference between all measures of all methods. But Weighted EMD and Reversible EMD perform slightly better than other methods (by around 2%).

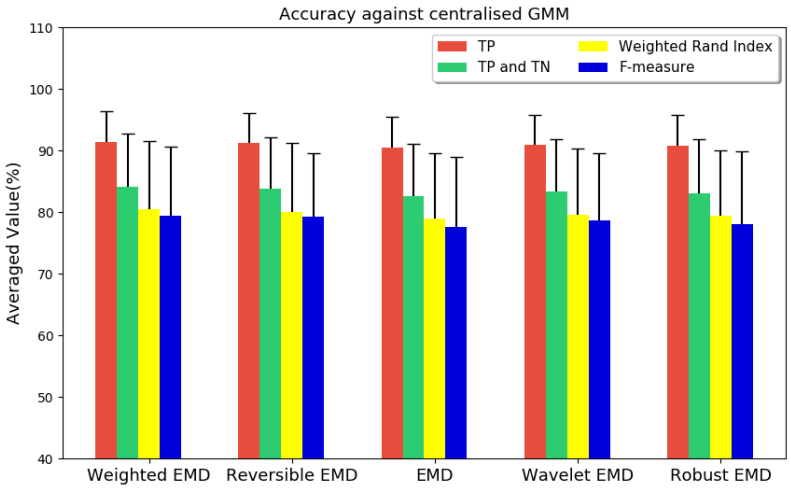


Figure 8.14: Accuracy against centralised GMM with fully symmetric Gaussians

Figure 8.15 shows that accuracy against ground truth. Again, Weighted EMD and Reversible EMD perform a little better than other methods.

Figure 8.16 shows that accuracy against pattern. Weighted EMD achieves higher accuracy than EMD. And both weighted EMD and EMD perform much better than other methods. wavelet EMD performs worst in all methods.

Figure 8.17 shows the accuracy against centralised GMM when uniformly distributed datasets are used. Compared that with fully symmetric Gaussians, the

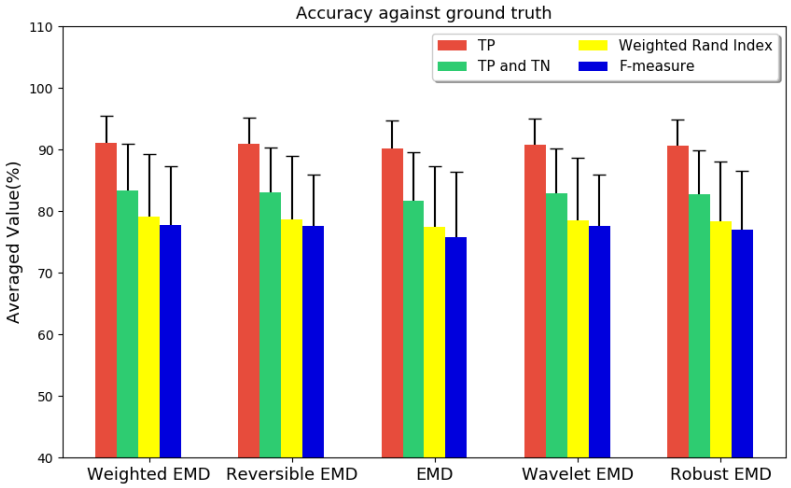


Figure 8.15: Accuracy against ground truth with fully symmetric Gaussians

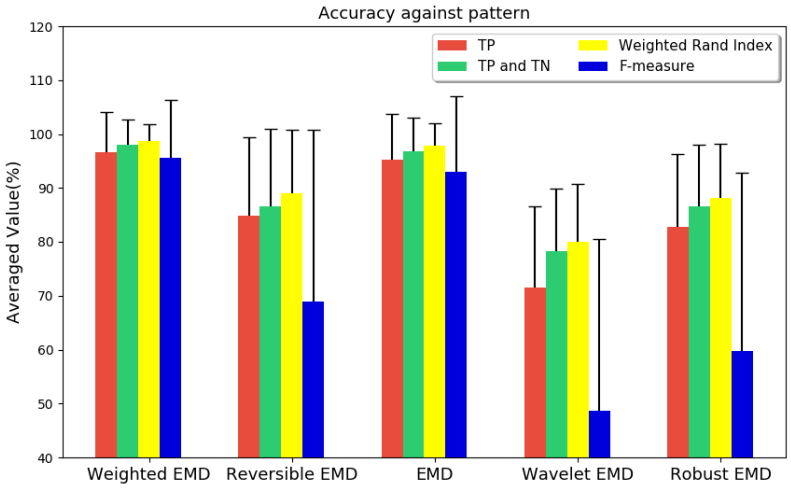


Figure 8.16: Accuracy against pattern with fully symmetric Gaussians

accuracy drops. We expected this since the clustering algorithm is not suitable for the underlying dataset. Similarly, accuracy against ground truth dropped as well (refer to figure 8.18). In terms of accuracy against pattern, EMD and weighted EMD perform much better. Note that there is little difference between EMD and weighted EMD. Of all methods, wavelet EMD and robust EMD perform the worst.

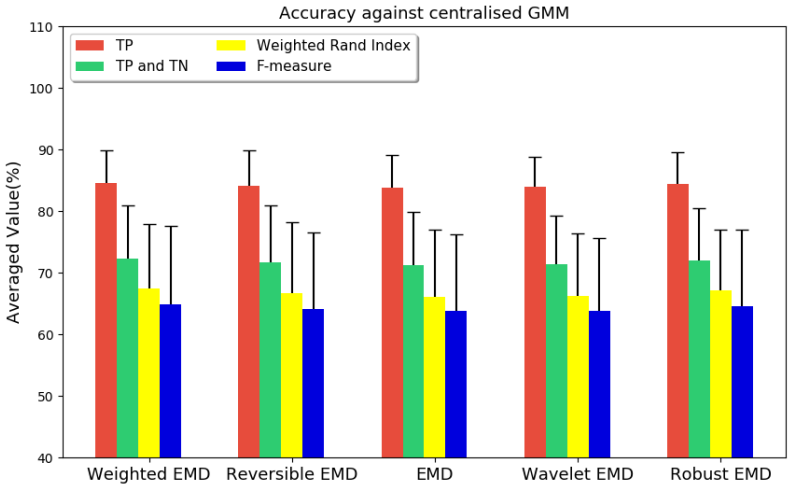


Figure 8.17: Accuracy against centralised GMM with datasets generated from uniform distributions

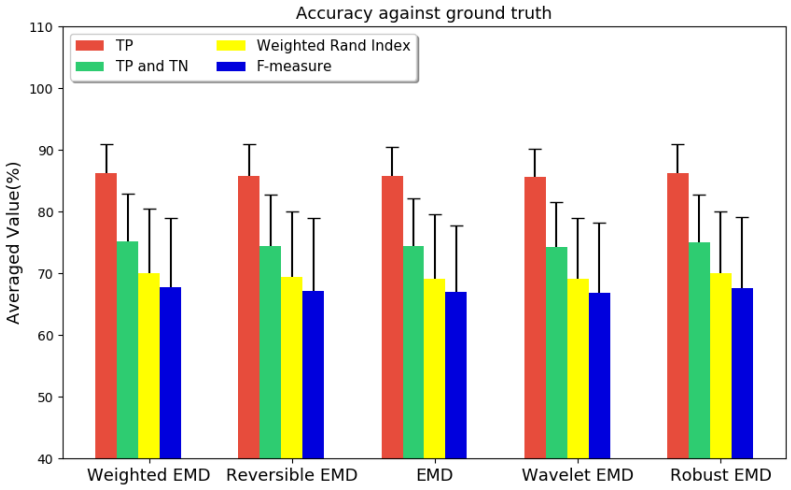


Figure 8.18: Accuracy against ground truth with uniformly distributed dataset

8.3.2 SG as clustering algorithm

In this section, we show the comparison result when SG is used as the clustering algorithm. As we expected, there is little difference between all measurements in terms of accuracy against centralised SG (refer to figure 8.20).

Figure 8.21 shows that, compared to the accuracy against centralised SG, the accuracy against ground truth drops. Again, all measurements follow the same pattern and there is little difference between them.

Regarding accuracy against pattern, weighted EMD performs slightly better than

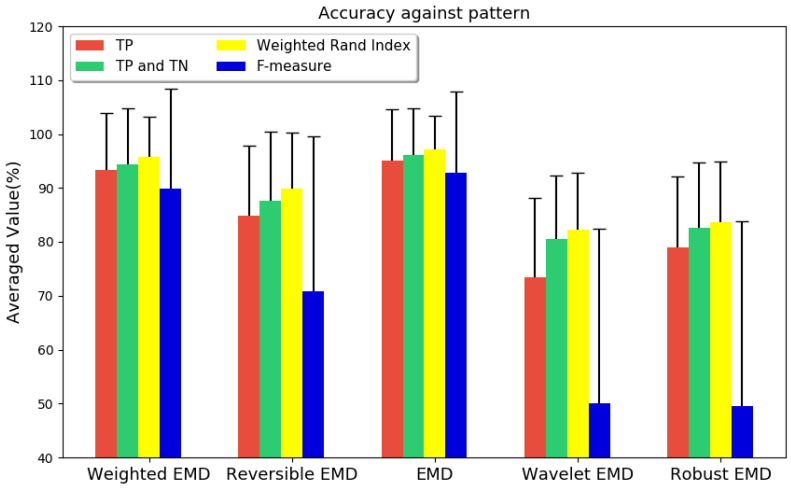


Figure 8.19: Accuracy against pattern with uniformly distributed dataset

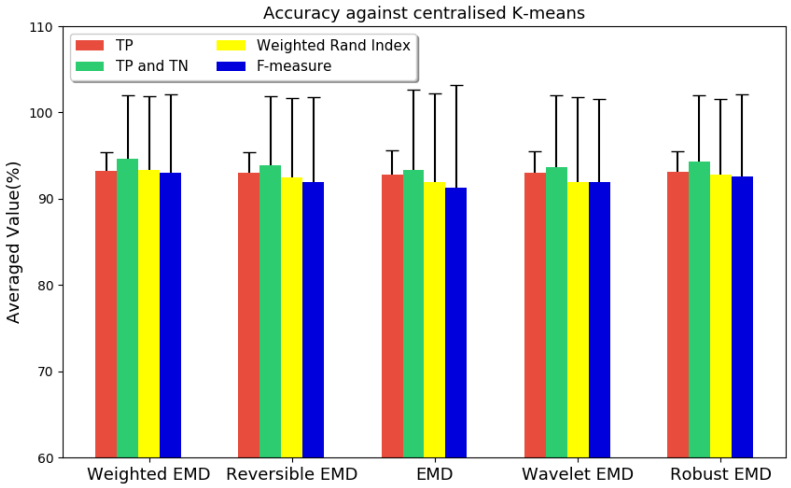


Figure 8.20: Accuracy against centralised SG with fully symmetric Gaussians

EMD (refer to figure 8.22). And both weighted EMD and EMD outperform the rest of methods.

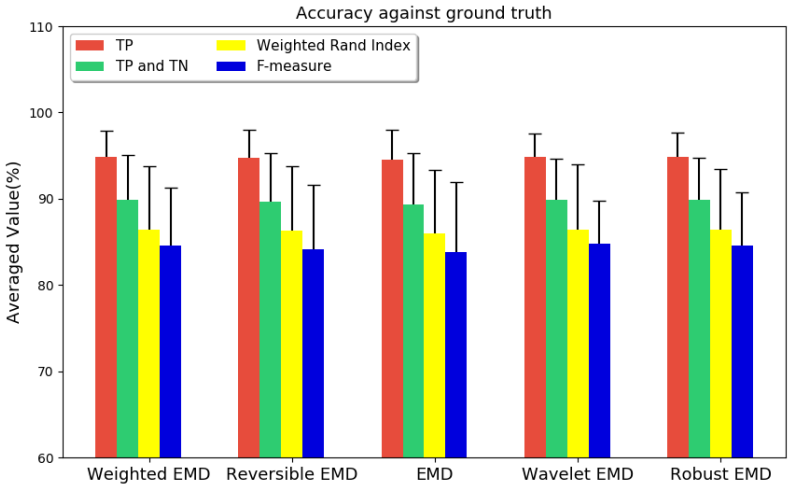


Figure 8.21: Accuracy against ground truth with fully symmetric Gaussians

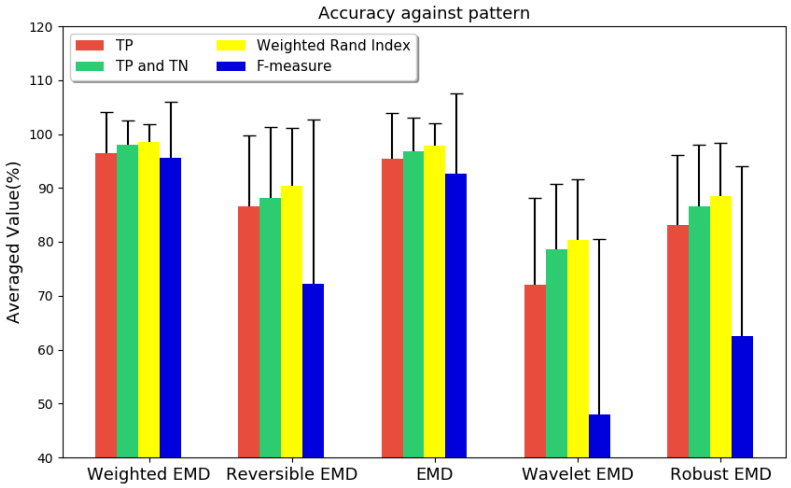


Figure 8.22: Accuracy against pattern with fully symmetric Gaussians

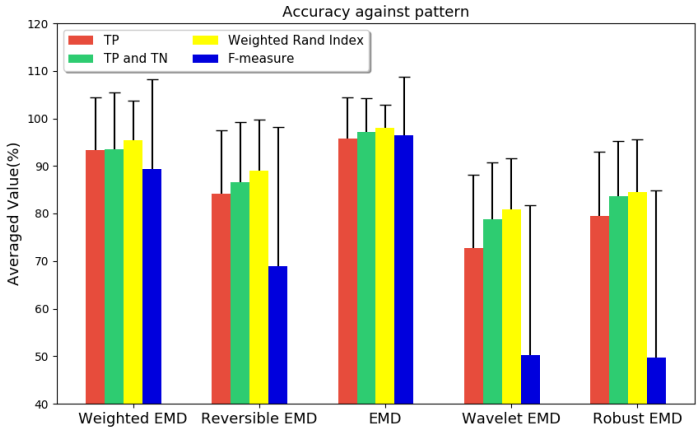


Figure 8.25: Accuracy against pattern with uniformly distributed dataset

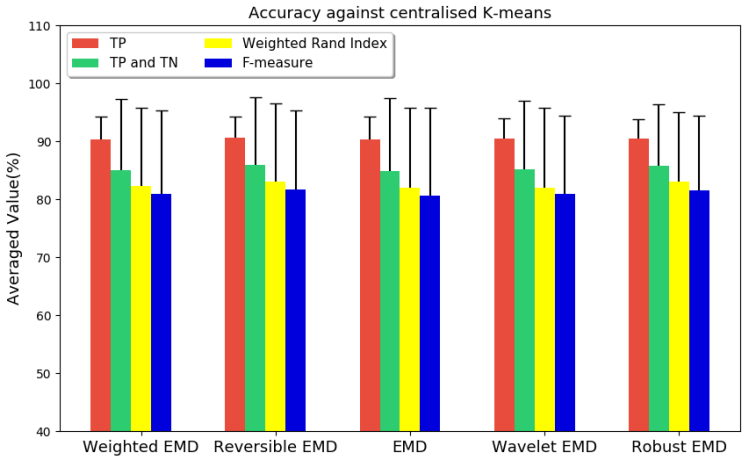


Figure 8.23: Accuracy against centralised SG with uniformly distributed dataset

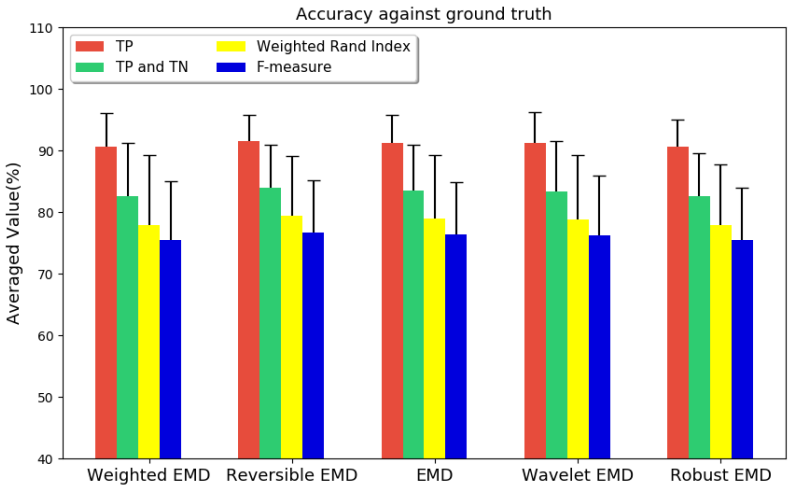


Figure 8.24: Accuracy against ground truth with uniformly distributed dataset

Figure 8.23, 8.24 and 8.25 show the accuracy against centralised SG, ground truth and pattern with uniformly distributed dataset, separately. Again, there are little differences between both accuracy measures on data points. Regarding the accuracy against pattern, EMD outperforms other methods, followed by weighted EMD. Wavelet EMD and robust EMD perform the worst in all methods.

8.4 Predict the sub-pattern assignment of agent

In this section, we show the comparison result on the two proposed methods (V1 and V2) to predict which sub-pattern the agent belongs to. When agents are not told the group assignment of all agents, agents have to compute the similarity between its cluster model and the cluster model inside the patterns. It could compare the cluster model with the cluster model of patterns (V1), or compare the raw data points with the cluster model of patterns (V2).

This experiment is tested on 10-agent network and repeated for 20 runs. The underlying datasets are generated from fully symmetric Gaussians and K-means is used as the clustering algorithm. We also compare the two different ways to build the similarity tables: build a symmetric table based on agent's prediction (denoted by MD) and build an asymmetric table that agent always go back to the models received and check who it is together with and apart from (MD1). In the experiment, we only focus on the three patterns networks (5:3:2).

Figure 8.26 shows the result when the ground truth distributions to generate data points in the second and third sub-patterns are similar. We expected that the estimated number of patterns are close to 2, since the cluster model by agents in these two sub-patterns are similar and it is difficult for the proposed algorithms to predict the same number of patterns as ground truth. It shows that comparing agent's basic models (V1), rather than raw data points (V2), with cluster model of patterns performs better. The most possible reason is that the local raw data point may not fit well with the learned final models.

Measures/datasets			Circled Gaussians (Slightly overlapped vs Heavily overlapped, 5:3:2, similar, MD1)										
			S1_A1_R1		*S1_A2_R1*		*S2_A1_R1*		*S1_A2_R2*	*S1_A1_R2*	*S2_A1_R2*	*C_R_R2*	*C_B_R2*
Convergence Time			V1	V2	V1	V2	V1	V2	6.45(0.45)	6.12(0.57)	7.99(0.88)	12.64(1.21)	6.66(0.7)
			6.04(0.33)	6.13(0.28)	8.78(1.69)	7.25(0.76)	7.3(0.23)	7.37(0.37)					
Information			37.5(10.1)	37.79(9.82)	43.64(10.13)	43.42(10.06)	72.18(11.3)	73.09(11.37)	44.92(10.01)	38.02(9.44)	72.07(11.41)	19.(42.45)	3.4(2.8)
E1	*C*	*TP*	85.98(4.24)	85.71(5.35)	82.85(3.18)	75.24(6.86)	82.67(3.15)	74.81(4.81)	82.49(3.38)	86.74(5.48)	82.55(2.81)	-------	95.31(3.48)
		TP+TN	89.23(3.56)	90.07(3.74)	87.63(2.33)	82.42(5.60)	87.51(2.22)	82.61(4.41)	87.01(2.35)	90.26(3.80)	87.76(1.86)		96.99(2.34)
	G	*TP*	77.83(3.18)	78.51(4.02)	74.62(0.77)	67.94(6.36)	74.7(0.62)	68.78(5.65)	74.89(0.58)	78.79(3.73)	74.6(0.72)	74.7(5.01)	88.56(1.42)
		TP+TN	82.43(2.24)	83.22(3.08)	80.05(0.56)	76.37(4.67)	80.19(0.34)	76.54(3.74)	80.31(0.51)	83.37(3.08)	80.29(0.46)	91.37(0.39)	91.28(1.15)
E2	*C*	*TP*	94.5(5.65)	92.99(7.45)	89.4(1.87)	93.6(7.05)	91.7(7.10)	91.8(7.6)	91.09(7.36)	95.59(4.45)	91.4(8.16)	-------	82.9(8.43)
		TP+TN	95.09(4.31)	94.85(4.24)	93.37(5.50)	94.48(4.89)	93.28(4.51)	94.06(4.03)	94.29(4.08)	95.27(3.63)	92.35(5.94)		90.11(4.82)
	G	*TP*	79.61(8.59)	80.27(7.44)	73.1(5.09)	76.0(9.35)	73.5(5.71)	74.4(5.27)	77.12(5.08)	80.51(7.96)	76.1(5.91)	84.0(4.5)	69.9(1.98)
		TP+TN	88.29(3.84)	88.52(3.56)	84.96(1.87)	86.37(4.66)	85.08(1.75)	85.09(1.83)	86.69(2.79)	88.61(3.53)	85.23(1.62)	89.17(3.18)	79.37(1.46)
E3	*P*	*TP*	100.0(0.0)	97.5(4.45)	98.6(4.20)	50.4(19.83)	99.6(1.2)	54.0(27.23)	100.0(0.0)	100.0(0.0)	100.0(0.0)	100.0(0.0)	100.0(0.0)
		TP+TN	100.0(0.0)	97.5(4.45)	98.6(4.20)	50.4(19.83)	99.6(1.2)	54.0(27.23)	100.0(0.0)	100.0(0.0)	100.0(0.0)	100.0(0.0)	100.0(0.0)
		TP	100.0(0.0)	97.5(4.45)	98.6(4.20)	50.4(19.83)	99.6(1.2)	54.0(27.23)	100.0(0.0)	100.0(0.0)	100.0(0.0)	100.0(0.0)	100.0(0.0)
		TP+TN	78.0(0.0)	76.5(3.38)	77.4(1.8)	58.5(5.61)	78.1(0.3)	56.7(9.96)	78.0(0.0)	78.0(0.0)	78.0(0.0)	78.0(0.0)	78.0(0.0)
Learned K([5,5,5])			3.43(0.34)	3.64(0.12)	3.5(0.0)	3.55(0.15)	3.59(0.12)	3.59(0.17)	3.56(0.07)	3.63(0.14)	3.6(0.09)	4.25(0.25)	3.5(0.0)
Learned Patterns			2.0(0.0)	2.0(0.0)	2.0(0.0)	2.0(0.0)	2.0(0.0)	2.0(0.0)	2.0(0.0)	2.0(0.0)	2.0(0.0)	2.0(0.0)	2.0(0.0)

Figure 8.26: Performance on datasets generated from fully symmetric Gaussians with K-means as clustering algorithm (1)

Figure 8.27 shows the result when the ground truth distributions to generate data points in the second and third sub-patterns are dissimilar instead. Since the

cluster models are dissimilar, we expected that the number of patterns are close to the ground truth number 3 (above 2.8). Again, the accuracy against pattern with method V1 is higher than that with V2.

Measures/datasets			Circled Gaussians (Slightly overlapped vs Heavily overlapped, 5:3:2, dissimilar, MD1)										
			S1_A1_R1		*S1_A2_R1*		*S2_A1_R1*		*S1_A2_R2*	*S1_A1_R2*	*S2_A1_R2*	*C_R_R2*	*C_B_R2*
Convergence Time			V1	V2	V1	V2	V1	V2	6.64(0.72)	6.28(0.42)	7.86(0.2)	16.27(2.46)	9.06(1.57)
			6.52(0.59)	6.31(0.45)	7.31(0.36)	7.79(1.17)	7.57(0.3)	8.61(0.96)					
Information			37.66(10.54)	37.64(9.81)	44.08(9.76)	44.2(10.69)	72.57(11.16)	73.09(11.26)	43.71(10.62)	37.72(9.84)	72.28(10.98)	19.5(42.45)	3.4(2.8)
E1	*C*	*TP*	91.1(4.85)	89.75(5.40)	87.35(7.31)	76.24(5.25)	87.09(7.49)	75.29(5.0)	86.26(6.57)	89.5(6.22)	87.43(4.62)	-------	95.6(2.64)
		TP+TN	93.73(3.24)	93.19(5.40)	90.72(4.92)	84.0(4.27)	90.19(5.90)	85.61(5.40)	89.94(4.70)	92.77(3.5)	90.72(2.96)		97.63(1.82)
	G	*TP*	83.08(3.18)	82.55(4.66)	80.58(3.77)	71.3(6.39)	79.37(4.68)	74.81(8.98)	79.24(2.90)	82.7(3.17)	79.23(2.68)	75.62(6.42)	88.84(1.61)
		TP+TN	86.97(2.56)	86.47(3.89)	85.01(3.10)	79.59(5.26)	83.8(4.06)	81.74(7.03)	83.9(2.26)	87.16(2.37)	83.81(2.18)	90.5(1.15)	91.55(1.28)
E2	*C*	*TP*	90.74(8.06)	93.05(6.46)	91.1(6.47)	91.7(8.03)	91.0(8.40)	92.8(5.98)	94.96(3.90)	92.74(6.42)	92.1(7.35)	-------	90.0(7.34)
		TP+TN	99.63(0.29)	99.42(0.95)	98.57(3.20)	98.67(3.61)	97.89(4.93)	99.77(0.40)	99.32(0.82)	99.45(0.68)	99.71(0.31)		93.92(3.88)
	G	*TP*	82.64(5.33)	80.62(6.60)	80.8(8.39)	82.5(9.76)	75.5(12.09)	78.7(12.30)	80.25(6.27)	81.91(7.87)	73.8(10.05)	88.7(2.99)	83.07(3.43)
		TP+TN	98.9(0.78)	98.2(2.51)	97.64(4.75)	97.67(4.92)	97.47(4.86)	99.23(0.61)	97.77(2.69)	98.26(2.18)	99.23(0.33)	92.06(2.29)	87.33(2.61)
E3	*P*	*TP*	100.0(0.0)	96.4(5.55)	99.3(2.10)	35.3(19.43)	100.0(0.0)	56.4(31.87)	100.0(0.0)	100.0(0.0)	100.0(0.0)	100.0(0.0)	100.0(0.0)
		TP+TN	100.0(0.0)	96.4(5.55)	99.3(2.10)	35.3(19.43)	100.0(0.0)	56.4(31.87)	100.0(0.0)	100.0(0.0)	100.0(0.0)	100.0(0.0)	100.0(0.0)
		TP	100.0(0.0)	96.4(5.55)	99.3(2.10)	35.3(19.43)	100.0(0.0)	56.4(31.87)	100.0(0.0)	100.0(0.0)	100.0(0.0)	100.0(0.0)	100.0(0.0)
		TP+TN	98.7(3.9)	97.2(4.02)	97.9(4.35)	58.0(11.35)	98.7(3.9)	70.6(14.64)	97.4(5.2)	98.7(3.9)	100.0(0.0)	98.7(3.9)	98.7(3.9)
Learned K([5,5,5])			3.89(0.59)	4.16(0.13)	4.1(0.15)	4.15(0.19)	4.15(0.09)	4.16(0.12)	4.16(0.2)	4.14(0.11)	4.11(0.06)	4.48(0.16)	4.18(0.19)
Learned Patterns			2.9(0.3)	2.9(0.3)	2.9(0.3)	2.8(0.4)	2.9(0.3)	2.9(0.3)	2.8(0.4)	2.9(0.3)	3.0(0.0)	2.9(0.3)	2.9(0.3)

Figure 8.27: Performance on datasets generated from fully symmetric Gaussians with K-means as clustering algorithm (2)

Figure 8.28 and 8.29 show the results when a symmetric table is built to measure the similarity between agents. Compared to figure 8.26, the accuracy against pattern in figure 8.28 is a little higher (see V1 of $S1_A2_R1$). That is because agents know the size of network and all agents finish with the same model. If there is only one pattern in the network and all agents infer the pattern assignment based on the same model, we expected that accuracy achieved by symmetric table is much accurate.

Similarly, when an asymmetric table is built (figure 8.27 and 8.29), the accuracy against pattern by V1 and V2 are similar and the only difference occurs in $S1_A2_R1$, where the accuracy against pattern by V1 in figure 8.29 is 0.8% higher. That is because it is not difficult for the proposed algorithms to predict the correct group assignment of all agents when the ground truth distributions to generate data points in different sub-patterns are distinct.

To summarise, when agents are not told the group assignment of all agents and have to predict its group assignment, measuring its cluster models (V1) with patterns received performs better. In addition, there is little difference in these two methods (less than 1% in accuracy against patterns).

Measures/datasets			Circled Gaussians (Slightly overlapped vs Heavily overlapped, 5:3:2, similar, MD)										
			S1_A1_R1		S1_A2_R1		S2_A1_R1		S1_A2_R2	S1_A1_R2	S2_A1_R2	C_R_R2	C_B_R2
Convergence Time			V1	V2	V1	V2	V1	V2	6.5(0.4)	6.12(0.57)	7.99(0.88)	12.93(1.43)	6.89(0.95)
			6.04(0.33)	6.13(0.28)	7.07(0.56)	6.75(0.8)	7.3(0.23)	7.37(0.37)					
Information			37.5(10.1)	37.79(9.82)	43.66(10.4)	44.01(10.5)	72.18(11.3)	73.09(11.37)	43.24(10.19)	38.02(9.44)	72.07(11.41)	19.5(42.45)	3.4(2.8)
E1	C	TP	85.98(4.24)	85.71(5.35)	84.45(5.63)	72.25(3.75)	82.67(3.15)	74.81(4.81)	82.62(3.28)	86.74(5.48)	82.55(2.81)	-------	95.87(3.54)
		TP+TN	89.23(3.56)	90.07(3.74)	88.19(4.51)	80.4(2.26)	87.51(2.22)	82.61(4.41)	86.85(2.37)	90.26(3.80)	87.76(1.86)		97.26(2.56)
	G	TP	77.83(3.18)	78.51(4.02)	75.93(3.89)	64.04(2.68)	74.7(0.62)	68.78(5.65)	74.77(0.70)	78.79(3.73)	74.6(0.72)	75.34(4.15)	88.52(1.66)
		TP+TN	82.43(2.24)	83.22(3.08)	81.38(3.09)	73.48(1.64)	80.19(0.34)	76.54(3.74)	80.2(0.38)	83.37(3.08)	80.29(0.46)	90.45(2.96)	91.31(1.50)
E2	C	TP	94.5(5.65)	92.99(7.45)	93.1(8.09)	92.6(6.79)	91.7(7.10)	91.8(7.6)	93.83(6.45)	95.59(4.45)	91.4(8.16)	-------	81.8(7.93)
		TP+TN	95.09(4.31)	94.85(4.24)	94.9(4.20)	93.82(5.33)	93.28(4.51)	94.06(4.03)	94.86(3.90)	95.27(3.63)	92.35(5.94)		90.35(4.03)
	G	TP	79.61(8.59)	80.27(7.44)	77.5(8.97)	76.4(9.11)	73.5(5.71)	74.4(5.27)	79.02(6.32)	80.51(7.96)	76.1(5.91)	83.0(5.91)	70.45(3.06)
		TP+TN	88.29(3.84)	88.52(3.56)	86.45(4.63)	86.73(4.33)	85.08(1.75)	85.09(1.83)	87.68(3.43)	88.61(3.53)	85.23(1.62)	88.69(3.73)	79.88(1.96)
E3	P	TP	100.0(0.0)	97.5(4.45)	100.0(0.0)	82.0(15.89)	99.6(1.2)	54.0(27.23)	100.0(0.0)	100.0(0.0)	100.0(0.0)	100.0(0.0)	100.0(0.0)
		TP+TN	100.0(0.0)	97.5(4.45)	100.0(0.0)	82.2(15.89)	99.6(1.2)	54.0(27.23)	100.0(0.0)	100.0(0.0)	100.0(0.0)	100.0(0.0)	100.0(0.0)
		TP	100.0(0.0)	97.5(4.45)	100.0(0.0)	82.0(15.89)	99.6(1.2)	54.0(27.23)	100.0(0.0)	100.0(0.0)	100.0(0.0)	100.0(0.0)	100.0(0.0)
		TP+TN	78.0(0.0)	76.5(3.38)	78.0(0.0)	42.2(12.08)	78.1(0.3)	56.7(9.96)	78.0(0.0)	78.0(0.0)	78.0(0.0)	78.0(0.0)	78.0(0.0)
Learned K([5,5,5])			3.43(0.34)	3.64(0.12)	3.55(0.15)	3.55(0.15)	3.59(0.12)	3.59(0.17)	3.58(0.07)	3.63(0.14)	3.6(0.09)	4.25(0.25)	3.55(0.15)
Learned Patterns			2.0(0.0)	2.0(0.0)	2.0(0.0)	2.0(0.0)	2.0(0.0)	2.0(0.0)	2.0(0.0)	2.0(0.0)	2.0(0.0)	2.0(0.0)	2.0(0.0)

Figure 8.28: Performance on datasets generated from fully symmetric Gaussians with K-means as clustering algorithm (3)

Measures/datasets			Circled Gaussians (Slightly overlapped vs Heavily overlapped, 5:3:2, dissimilar, MD)										
			S1_A1_R1		S1_A2_R1		S2_A1_R1		S1_A2_R2	S1_A1_R2	S2_A1_R2	C_R_R2	C_B_R2
Convergence Time			V1	V2	V1	V2	V1	V2	6.52(0.62)	6.28(0.42)	7.86(0.2)	11.48(0.48)	7.45(0.51)
			6.52(0.59)	6.31(0.45)	6.84(0.51)	6.96(0.86)	7.57(0.3)	8.61(0.96)					
Information			37.66(10.54)	37.64(9.81)	43.55(9.34)	43.99(10.25)	72.57(11.16)	73.09(11.26)	44.86(10.71)	37.72(9.84)	72.28(10.98)	19.5(42.45)	3.4(2.8)
E1	C	TP	91.1(4.85)	89.75(5.40)	87.68(5.94)	76.2(6.21)	87.09(7.49)	75.29(5.0)	83.86(5.85)	89.5(6.22)	87.43(4.62)	-------	95.49(3.03)
		TP+TN	93.73(3.24)	93.19(5.40)	90.1(4.62)	85.47(4.96)	90.19(5.90)	85.61(5.40)	87.29(4.58)	92.77(3.5)	90.72(2.96)		97.26(2.14)
	G	TP	83.08(3.18)	82.55(4.66)	79.86(4.14)	73.6(7.69)	79.37(4.68)	74.81(8.98)	77.13(2.54)	82.7(3.17)	79.23(2.68)	72.88(5.73)	88.81(1.62)
		TP+TN	86.97(2.56)	86.47(3.89)	84.26(3.52)	81.28(5.83)	83.8(4.06)	81.74(7.03)	81.75(2.40)	87.16(2.37)	83.81(2.18)	89.75(2.53)	91.38(1.29)
E2	C	TP	90.74(8.06)	93.05(6.46)	94.5(5.61)	92.0(7.94)	91.0(8.40)	92.8(5.98)	91.35(7.62)	92.74(6.42)	92.1(7.35)	-------	93.78(3.99)
		TP+TN	99.63(0.29)	99.42(0.95)	99.31(0.90)	99.78(0.33)	97.89(4.93)	99.77(0.40)	98.5(2.70)	99.45(0.68)	99.71(0.31)		95.92(2.48)
	G	TP	82.64(5.33)	80.62(6.60)	75.7(11.38)	80.6(12.53)	75.5(12.09)	78.7(12.30)	81.99(3.23)	81.91(7.87)	73.8(10.05)	88.88(2.91)	82.82(3.97)
		TP+TN	98.9(0.78)	98.2(2.51)	96.1(5.90)	97.87(4.73)	97.47(4.86)	99.23(0.61)	97.52(3.11)	98.26(2.18)	99.23(0.33)		87.3(2.96)
E3	M2	TP	100.0(0.0)	96.4(5.55)	100.0(0.0)	73.5(19.98)	100.0(0.0)	56.4(31.87)	100.0(0.0)	100.0(0.0)	100.0(0.0)	100.0(0.0)	100.0(0.0)
		TP+TN	100.0(0.0)	96.4(5.55)	100.0(0.0)	73.5(19.98)	100.0(0.0)	56.4(31.87)	100.0(0.0)	100.0(0.0)	100.0(0.0)	100.0(0.0)	100.0(0.0)
		TP	100.0(0.0)	96.4(5.55)	100.0(0.0)	73.5(19.98)	100.0(0.0)	56.4(31.87)	100.0(0.0)	100.0(0.0)	100.0(0.0)	100.0(0.0)	100.0(0.0)
		TP+TN	98.7(3.9)	97.2(4.02)	98.7(3.9)	52.0(16.12)	98.7(3.9)	70.6(14.64)	97.4(5.2)	98.7(3.9)	100.0(0.0)	97.4(5.2)	98.7(3.9)
Learned K([5,5,5])			3.89(0.59)	4.16(0.13)	4.02(0.24)	4.07(0.13)	4.15(0.09)	4.16(0.12)	4.14(0.17)	4.14(0.11)	4.11(0.06)	4.5(0.15)	4.18(0.19)
Learned Patterns			2.9(0.3)	2.9(0.3)	2.9(0.3)	2.9(0.3)	2.9(0.3)	2.9(0.3)	2.8(0.4)	2.9(0.3)	3.0(0.0)	2.8(0.4)	2.9(0.3)

Figure 8.29: Performance on datasets generated from fully symmetric Gaussians with K-means as clustering algorithm (4)

8.5 Comparison result on multiple sparse networks

In this section, we show the experimental results on sparse networks. Similarly, three different clustering algorithms are used to cluster the data points that generated from fully symmetric Gaussians, partially symmetric Gaussians and uniform distributions. The main difference between dense network and sparse network lies in the amount of transmitted information. Compared to that with dense graph, fewer information is required.

8.5.1 K-means as clustering algorithm

When the network is sparse, figure 8.30, 8.31 and 8.32 show the experimental result with data points generated from fully symmetric Gaussians, partially symmetric Gaussians and uniformly distributed dataset, separately. When data

points are sampled from fully symmetric Gaussians, again, there is little difference in the number of patterns with different number of clusters when there are only two patterns. When there are three patterns (the fourth column), the number of patterns by the same number of clusters is underestimated, but the estimated number of patterns by different number of clusters is close to the ground truth patterns (see the horizontal dashed line). That is because it is difficult for algorithm to separate two similar clusters apart and it merges these two clusters into a big cluster.

When the underlying datasets are generated from partially symmetric Gaussians, the number of patterns predicted by different number of clusters is perfect when there are only two patterns. However, both algorithms with the same and different number of clusters are struggling to learn the correct number of patterns when there are three patterns.

Figure 8.32 shows the result with underlying datasets generated from uniformly distributions. Again, the number of patterns estimated by the same number of clusters is overestimated except the case of three patterns (5:3:2). However, when there are three patterns, the number of patterns estimated by the same number of clusters is much close to the ground truth. So we expected that the corresponding accuracy against pattern is much higher (refer to the last four rows). Note that the accuracy again pattern is the worst for the 9:1 split (the third column), where the accuracy against pattern is around 48%.

8.5.2 GMM as clustering algorithm

Figure 8.33 shows the result when datasets that generated from fully symmetric Gaussians are tested. The results show the same behaviour.

When the underlying datasets are generated from partially symmetric Gaussians (refer to figure 8.34), the results are similar to that with K-means as clustering algorithm.

Figure 8.35 shows the result when underlying datasets are generated from uniform distributions. As before, the results show the same pattern.

8.5.3 SG as clustering algorithm

Figure 8.36, 8.37 and 8.38 show the experimental result with datasets that generated from fully symmetric Gaussians, partially symmetric Gaussians and uniform distributions, separately. Again, the results show the same behaviour.



Figure 8.30: K-means on sparse network with multiple patterns and fully symmetric multi-dimensional Gaussians



Figure 8.31: K-means on sparse network with multiple patterns and partially symmetric multi-dimensional Gaussians

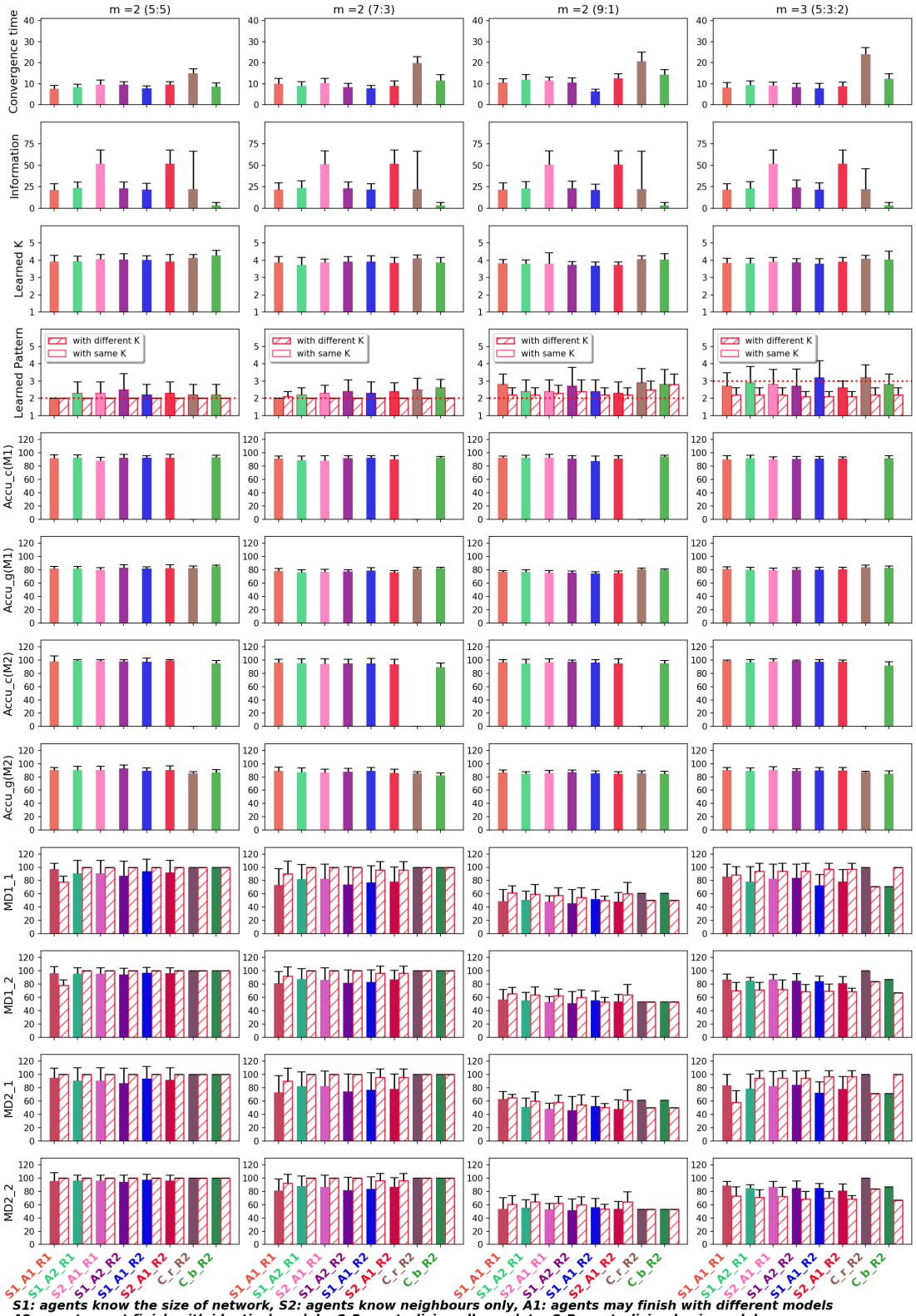


Figure 8.32: K-means on sparse network with multiple patterns and uniformly distributed dataset



Figure 8.33: GMM on sparse network with multiple patterns and fully symmetric multi-dimensional Gaussians



Figure 8.34: GMM on sparse network with multiple patterns and partially symmetric multi-dimensional Gaussians

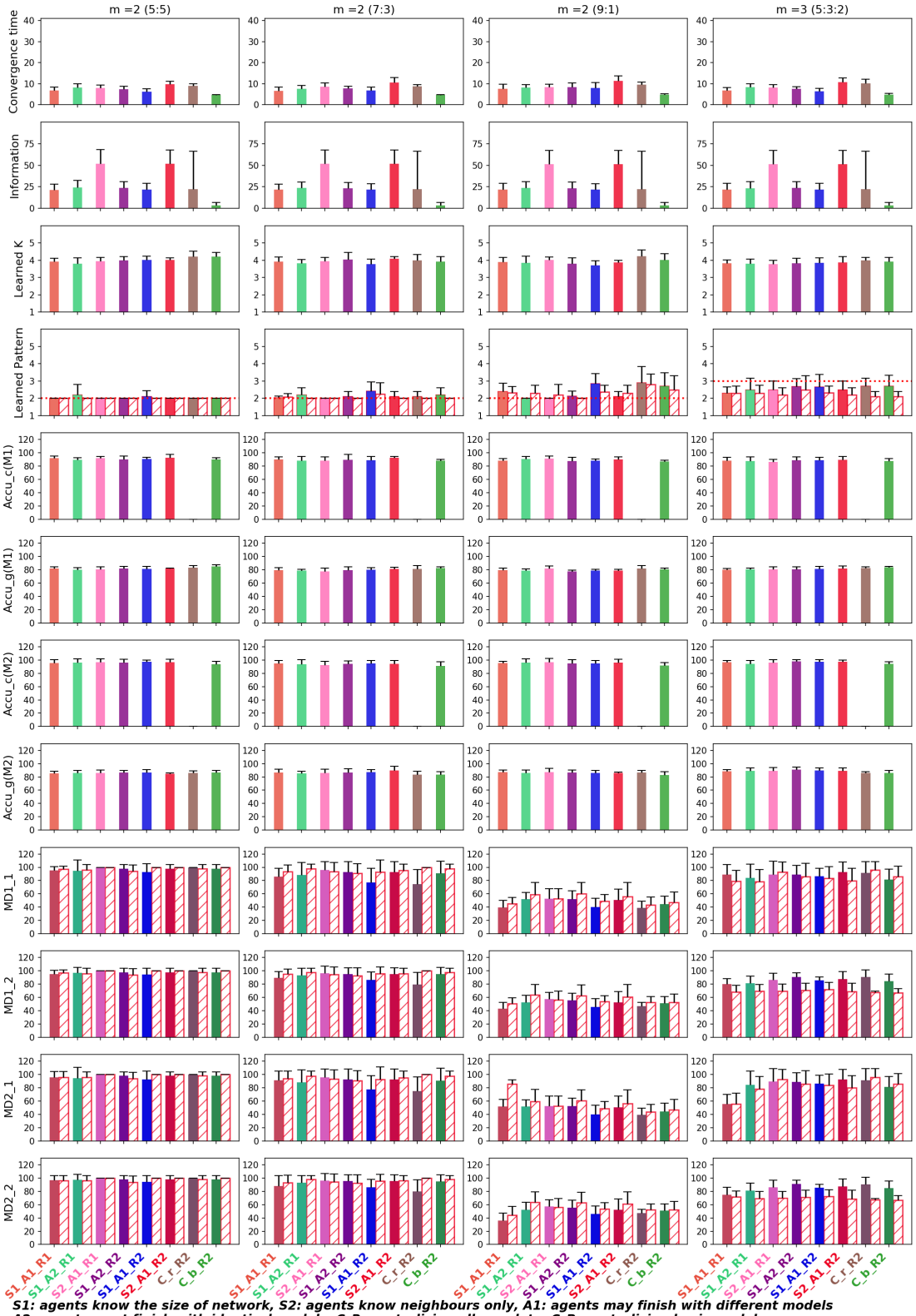


Figure 8.35: GMM on sparse network with multiple patterns and uniformly distributed dataset

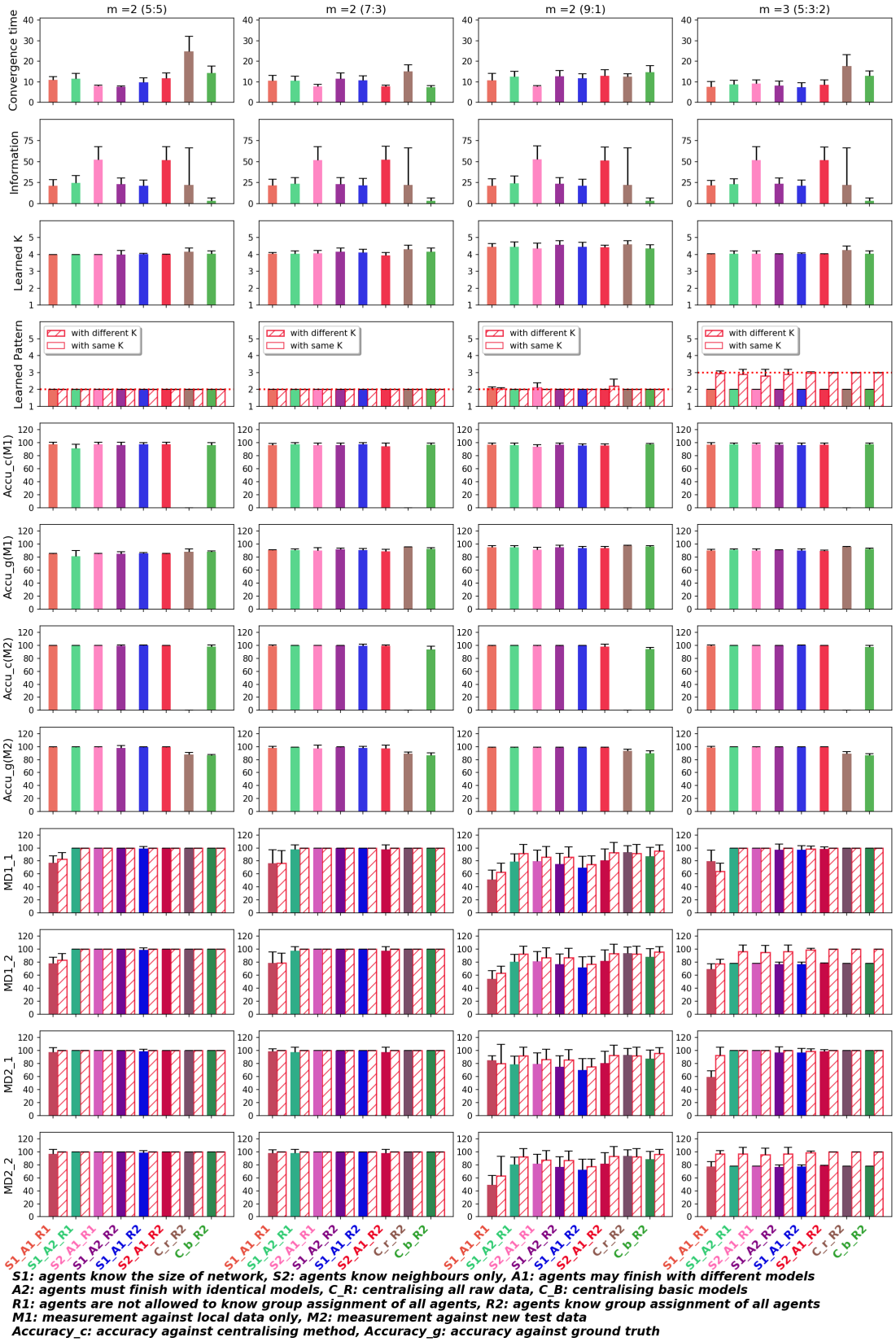


Figure 8.36: SG on sparse network with multiple patterns and fully symmetric multi-dimensional Gaussians



Figure 8.37: SG on sparse network with multiple patterns and partially symmetric multi-dimensional Gaussians

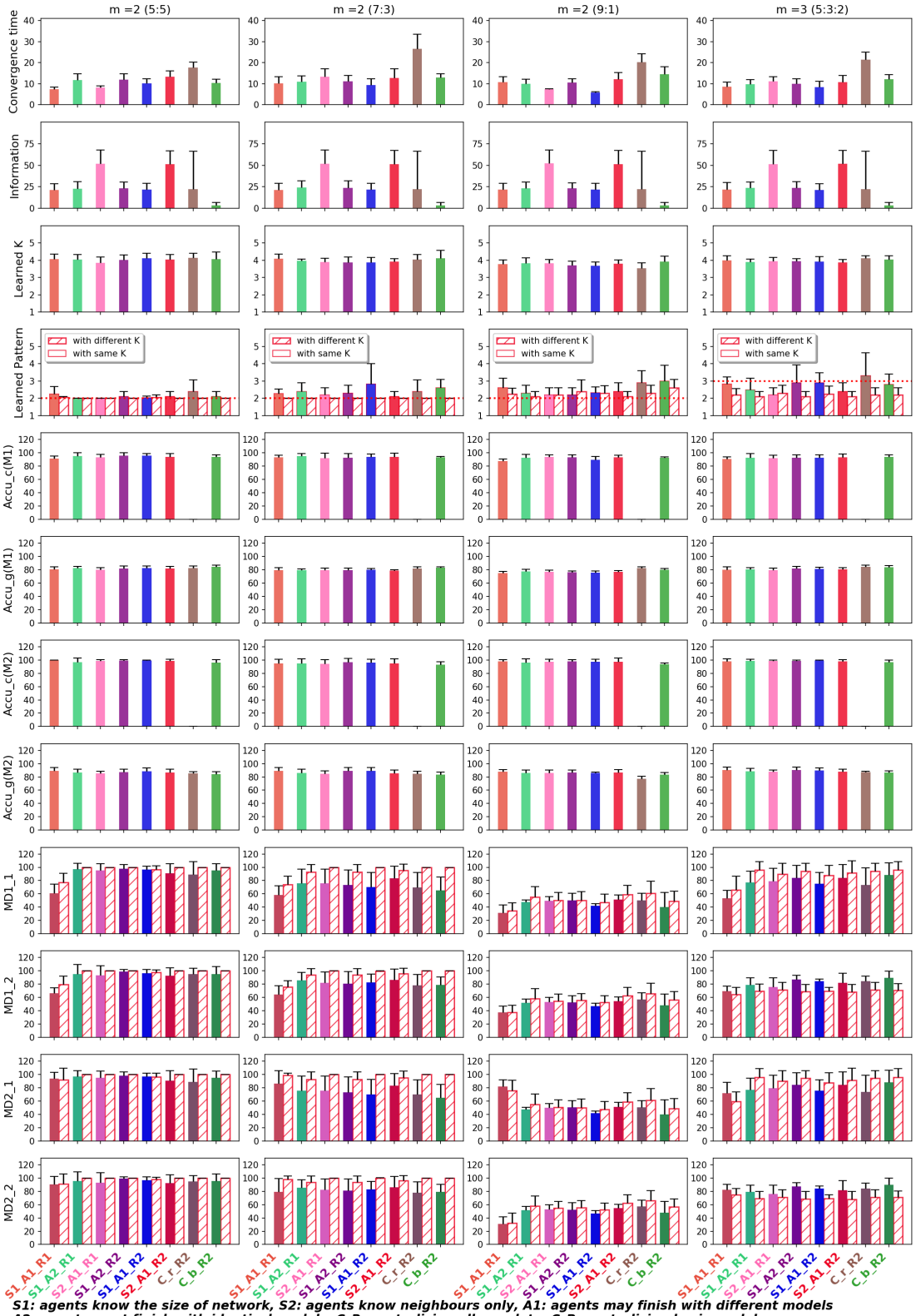


Figure 8.38: SG on sparse network with multiple patterns and uniformly distributed dataset

References

Ajay Agarwal and Minakshi Chauhan. Similarity measures used in recommender systems: a study. *International Journal of Engineering Technology Science and Research IJETSR, ISSN*, pages 2394–3386, 2017.

Ian F Akyildiz, Xudong Wang, and Weilin Wang. Wireless mesh networks: a survey. *Computer networks*, 47(4):445–487, 2005.

Shaker Alanazi, Kashif Saleem, Jalal Al-Muhtadi, and Abdelouahid Derhab. Analysis of denial of service impact on data routing in mobile ehealth wireless mesh network. *Mobile Information Systems*, 2016.

Douglas G Altman and J Martin Bland. How to obtain the p value from a confidence interval. *Bmj*, 343:d2304, 2011.

David Anderson and Kenneth Burnham. Aic myths and misunderstandings. *Website, April*, 2006.

Olatz Arbelaitz, Ibai Gurrutxaga, Javier Muguerza, JesúS M PéRez, and IñIgo Perona. An extensive comparative study of cluster validity indices. *Pattern Recognition*, 46(1):243–256, 2013.

David Arthur and Sergei Vassilvitskii. How slow is the k-means method? In *Proceedings of the twenty-second annual symposium on Computational geometry*, pages 144–153, 2006.

Olivier Bachem, Mario Lucic, S Hamed Hassani, and Andreas Krause. Approximate k-means++ in sublinear time. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 30, 2016.

Seema Bandyopadhyay and Edward J Coyle. An energy efficient hierarchical clustering algorithm for wireless sensor networks. In *IEEE INFOCOM 2003. Twenty-second Annual Joint Conference of the IEEE Computer and Communications Societies*, volume 3, pages 1713–1723. IEEE, 2003.

Sugato Basu, Ian Davidson, and Kiri Wagstaff. *Constrained clustering: Advances in algorithms, theory, and applications*. CRC Press, 2008.

Malika Bendechache, Nhien-An Le-Khac, and M-Tahar Kechadi. Hierarchical aggregation approach for distributed clustering of spatial datasets. In *2016 IEEE 16th International Conference on Data Mining Workshops (ICDMW)*, pages 1098–1103. IEEE, 2016.

Florence Bénézit, Vincent Blondel, Patrick Thiran, John Tsitsiklis, and Martin Vetterli. Weighted gossip: Distributed averaging using non-doubly stochastic matrices. In *2010 IEEE International Symposium on Information Theory*, pages 1753–1757. IEEE, 2010.

Derya Birant and Alp Kut. St-dbscan: An algorithm for clustering spatial–temporal data. *Data & knowledge engineering*, 60(1):208–221, 2007.

Ken Birman. The promise, and limitations, of gossip protocols. *ACM SIGOPS Operating Systems Review*, 41(5):8–13, 2007.

Sung-Hyuk Cha. Comprehensive survey on distance/similarity measures between probability density functions. *City*, 1(2):1, 2007.

Pierre Chainais and Cédric Richard. Learning a common dictionary over a sensor network. In *2013 5th IEEE International Workshop on Computational Advances in Multi-Sensor Adaptive Processing (CAMSAP)*, pages 133–136. IEEE, 2013.

Jianshu Chen, Zaid J Towfic, and Ali H Sayed. Dictionary learning over distributed models. *IEEE Transactions on Signal Processing*, 63(4):1001–1016, 2014.

James C Corbett, Jeffrey Dean, Michael Epstein, Andrew Fikes, Christopher Frost, Jeffrey John Furman, Sanjay Ghemawat, Andrey Gubarev, Christopher Heiser, Peter Hochschild, et al. Spanner: Google's globally distributed database. *ACM Transactions on Computer Systems (TOCS)*, 31(3):1–22, 2013.

Amir Daneshmand, Ying Sun, Gesualdo Scutari, Francisco Facchinei, and Brian M Sadler. Decentralized dictionary learning over time-varying digraphs. *Journal of Machine Learning Research*, 20(139):1–62, 2019.

Souptik Datta, Chris Giannella, and Hillol Kargupta. Approximate distributed k-means clustering over a peer-to-peer network. *IEEE Transactions on Knowledge and Data Engineering*, 21(10):1372–1388, 2008.

Ashwin D'Costa, Vinod Ramachandran, and Akbar M Sayeed. Distributed classification of gaussian space-time sources in wireless sensor networks. *IEEE Journal on Selected Areas in Communications*, 22(6):1026–1036, 2004.

Giuseppe Di Fatta, Francesco Blasa, Simone Cafiero, and Giancarlo Fortino. Fault tolerant decentralised k-means clustering for asynchronous large-scale networks. *Journal of Parallel and Distributed Computing*, 73(3):317–329, 2013.

Min Ding, Xiuzhen Cheng, and Guoliang Xue. Aggregation tree construction in sensor networks. In *2003 IEEE 58th Vehicular Technology Conference. VTC 2003-Fall*, volume 4, pages 2168–2172. IEEE, 2003.

M.E Dyer and A.M Frieze. A simple heuristic for the p-centre problem. *Operations Research Letters*, 3(6):285 – 288, 1985. ISSN 0167-6377. doi: https://doi.org/10.1016/0167-6377(85)90002-1. URL `http://www.sciencedirect.com/science/article/pii/0167637785900021`.

P Erdös and A Rényi. On random graphs i. *Publ. math. debrecen*, 6(290-297):18, 1959.

Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu, et al. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Kdd*, volume 96, pages 226–231, 1996.

Brendan J Frey and Delbert Dueck. Clustering by passing messages between data points. *science*, 315(5814):972–976, 2007.

Asghar Ghasemi and Saleh Zahediasl. Normality tests for statistical analysis: a guide for non-statisticians. *International journal of endocrinology and metabolism*, 10(2):486, 2012.

Arvind Giridhar and Praveen R Kumar. Computing and communicating functions over sensor networks. *IEEE Journal on selected areas in communications*, 23 (4):755–764, 2005.

Stewart Greenhill and Svetha Venkatesh. Distributed query processing for mobile surveillance. In *Proceedings of the 15th ACM international conference on Multimedia*, pages 413–422, 2007.

Tina Gui, Christopher Ma, Feng Wang, Jinyang Li, and Dawn E Wilkins. A novel cluster-based routing protocol wireless sensor networks using spider monkey optimization. In *IECON 2016-42nd Annual Conference of the IEEE Industrial Electronics Society*, pages 5657–5662. IEEE, 2016.

Guo-Dong Guo, Anil K Jain, Wei-Ying Ma, and Hong-Jiang Zhang. Learning similarity measure for natural image retrieval with relevance feedback. *IEEE Transactions on Neural Networks*, 13(4):811–820, 2002.

Greg Hamerly and Charles Elkan. Learning the k in k-means. In *Advances in neural information processing systems*, pages 281–288, 2004.

Eui-Hong Han and George Karypis. Feature-based recommendation system. In *Proceedings of the 14th ACM international conference on Information and knowledge management*, pages 446–452, 2005.

Jiawei Han, Jian Pei, and Micheline Kamber. *Data mining: concepts and techniques*. Elsevier, 2011.

Hassan Harb, Abdallah Makhoul, Samar Tawbi, and Raphaël Couturier. Comparison of different data aggregation techniques in distributed sensor networks. *IEEE Access*, 5:4250–4263, 2017.

Nils-Bastian Heidenreich, Anja Schindler, and Stefan Sperlich. Bandwidth selection for kernel density estimation: a review of fully automatic selectors. *AStA Advances in Statistical Analysis*, 97(4):403–433, 2013.

Wendi B Heinzelman, Anantha P Chandrakasan, and Hari Balakrishnan. An application-specific protocol architecture for wireless microsensor networks. *IEEE Transactions on wireless communications*, 1(4):660–670, 2002.

Wendi Rabiner Heinzelman, Joanna Kulik, and Hari Balakrishnan. Adaptive protocols for information dissemination in wireless sensor networks. In *Proceedings of the 5th annual ACM/IEEE international conference on Mobile computing and networking*, pages 174–185, 1999.

James Hensman, Nicolò Fusi, and Neil D Lawrence. Gaussian processes for big data. In *Proceedings of the Twenty-Ninth Conference on Uncertainty in Artificial Intelligence*, pages 282–290, 2013.

Frank L Hitchcock. The distribution of a product from several sources to numerous localities. *Journal of mathematics and physics*, 20(1-4):224–230, 1941.

Tang Hong, Zhu Liehuang, Cao Yuanda, and Wang Dazhen. A novel tree-based authenticated dynamic group key agreement protocol for wireless sensor network. In *2008 International Symposium on Electronic Commerce and Security*, pages 540–544. IEEE, 2008.

Anil K Jain and Richard C Dubes. *Algorithms for clustering data*. Prentice-Hall, Inc., 1988.

J Jetcheva. A simple protocol for multicast and broadcast in mobile ad hoc networks. *Internet Draft: draft-ietf-manetsimple-mbcast-01. txt*, 2001.

Ruoyi Jiang, Hongliang Fei, and Jun Huan. Anomaly localization for network data streams with graph joint sparse pca. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 886–894, 2011.

Iain M Johnstone and Arthur Yu Lu. Sparse principal components analysis. *arXiv preprint arXiv:0901.4392*, 2009.

Oded Kariv and S Louis Hakimi. An algorithmic approach to network location problems. i: The p-centers. *SIAM Journal on Applied Mathematics*, 37(3): 513–538, 1979.

Leonard Kaufman and Peter J Rousseeuw. *Finding groups in data: an introduction to cluster analysis*, volume 344. John Wiley & Sons, 2009.

Dinesh Kumar Kotary and Satyasai Jagannath Nandal. A point symmetry distance based k-means algorithm for distributed clustering in peer to peer networks. In *2019 IEEE International Conference on Systems, Man and Cybernetics (SMC)*, pages 3573–3579. IEEE, 2019.

L Krishnamachari, Deborah Estrin, and Stephen Wicker. The impact of data aggregation in wireless sensor networks. In *Proceedings 22nd international conference on distributed computing systems workshops*, pages 575–578. IEEE, 2002.

Luc Le Magoarou and Rémi Gribonval. Chasing butterflies: In search of efficient dictionaries. In *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 3287–3291. IEEE, 2015.

Sangseok Lee and Dong Kyu Lee. What is the proper way to apply the multiple comparison test? *Korean journal of anesthesiology*, 71(5):353, 2018.

Chiu-Kuo Liang, Yu-Jie Huang, and Jian-Da Lin. An energy efficient routing scheme in wireless sensor networks. In *22nd International Conference on Advanced Information Networking and Applications-Workshops (aina workshops 2008)*, pages 916–921. IEEE, 2008.

Aristidis Likas, Nikos Vlassis, and Jakob J Verbeek. The global k-means clustering algorithm. *Pattern recognition*, 36(2):451–461, 2003.

Peng Lin, Yinghui Wang, Hongsheng Qi, and Yiguang Hong. Distributed consensus-based k-means algorithm in switching multi-agent networks. *Journal of Systems Science and Complexity*, 31(5):1128–1145, 2018.

Stehpanie Lindsay, Cauligi S Raghavendra, and Krishna M Sivalingam. Data gathering in sensor networks using the energy delay metric. In *Proceedings of the 15th International Parallel & Distributed Processing Symposium*, page 188, 2001.

Julien Mairal, Jean Ponce, Guillermo Sapiro, Andrew Zisserman, and Francis R Bach. Supervised dictionary learning. In *Advances in neural information processing systems*, pages 1033–1040, 2009.

Julien Mairal, Francis Bach, Jean Ponce, and Guillermo Sapiro. Online learning for matrix factorization and sparse coding. *Journal of Machine Learning Research*, 11(1), 2010.

Matthew J Miller and Nitin H Vaidya. A mac protocol to reduce sensor network energy consumption using a wakeup radio. *IEEE Transactions on mobile Computing*, 4(3):228–242, 2005.

Todd K Moon. The expectation-maximization algorithm. *IEEE Signal processing magazine*, 13(6):47–60, 1996.

Xuanlong Nguyen, Michael I Jordan, and Bruno Sinopoli. A kernel-based learning approach to ad hoc sensor network localization. *ACM Transactions on Sensor Networks (TOSN)*, 1(1):134–152, 2005.

Vidhi S Patel and Chandresh R Parekh. Survey on sensor protocol for information via negotiation (spin) protocol. *IJRET: International Journal of Research in Engineering and Technology*, 3(03), 2014.

Ofir Pele and Michael Werman. A linear time histogram metric for improved sift matching. In *European conference on computer vision*, pages 495–508. Springer, 2008.

Ofir Pele and Michael Werman. Fast and robust earth mover's distances. In *2009 IEEE 12th International Conference on Computer Vision*, pages 460–467. IEEE, 2009.

Diego Peteiro-Barral and Bertha Guijarro-Berdiñas. A survey of methods for distributed machine learning. *Progress in Artificial Intelligence*, 2(1):1–11, 2013.

Rafael Coimbra Pinto and Paulo Martins Engel. A fast incremental gaussian mixture model. *PloS one*, 10(10):e0139931, 2015.

Joel B Predd, Sanjeev B Kulkarni, and H Vincent Poor. Distributed learning in wireless sensor networks. *IEEE Signal Processing Magazine*, 23(4):56–69, 2006.

Ramesh Rajagopalan and Pramod K Varshney. Data aggregation techniques in sensor networks: A survey. 2006.

William M Rand. Objective criteria for the evaluation of clustering methods. *Journal of the American Statistical association*, 66(336):846–850, 1971.

Yossi Rubner, Carlo Tomasi, and Leonidas J Guibas. A metric for distributions with applications to image databases. In *Sixth International Conference on Computer Vision (IEEE Cat. No. 98CH36271)*, pages 59–66. IEEE, 1998.

Yossi Rubner, Carlo Tomasi, and Leonidas J Guibas. The earth mover's distance as a metric for image retrieval. *International journal of computer vision*, 40(2): 99–121, 2000.

Harshavardhan Sabbineni and Krishnendu Chakrabarty. Location-aided flooding: an energy-efficient data dissemination protocol for wireless-sensor networks. *IEEE transactions on computers*, 54(1):36–46, 2005.

Yutaka Sasaki et al. The truth of the f-measure, 2007.

Simone Scardapane, Rosa Altilio, Massimo Panella, and Aurelio Uncini. Distributed spectral clustering based on euclidean distance matrix completion. In *2016 International Joint Conference on Neural Networks (IJCNN)*, pages 3093–3100. IEEE, 2016.

David W Scott. *Multivariate density estimation: theory, practice, and visualization*. John Wiley & Sons, 2015.

Alok Sharma and Kuldip K Paliwal. Fast principal component analysis using fixed-point algorithm. *Pattern Recognition Letters*, 28(10):1151–1155, 2007.

Simon J Sheather and Michael C Jones. A reliable data-based bandwidth selection method for kernel density estimation. *Journal of the Royal Statistical Society: Series B (Methodological)*, 53(3):683–690, 1991.

Haipeng Shen and Jianhua Z Huang. Sparse principal component analysis via

regularized low rank matrix approximation. *Journal of multivariate analysis*, 99(6):1015–1034, 2008.

Roger N Shepard. Toward a universal law of generalization for psychological science. *Science*, 237(4820):1317–1323, 1987.

Sameer Shirdhonkar and David W Jacobs. Approximate earth mover?s distance in linear time. In *2008 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8. IEEE, 2008.

Bernard W Silverman. *Density estimation for statistics and data analysis*, volume 26. CRC press, 1986.

Katayoun Sohrabi and Gregory J Pottie. Performance of a novel self-organization protocol for wireless ad-hoc sensor networks. In *Gateway to 21st Century Communications Village. VTC 1999-Fall. IEEE VTS 50th Vehicular Technology Conference*, volume 2, pages 1222–1226. IEEE, 1999.

Hüseyin Özgür Tan and Ibrahim Körpeolu. Power efficient data gathering and aggregation in wireless sensor networks. *ACM Sigmod Record*, 32(4):66–71, 2003.

Ying-Yao Ting, Chi-Wei Hsiao, and Huan-Sheng Wang. A data fusion-based fire detection system. *IEICE TRANSACTIONS on Information and Systems*, 101 (4):977–984, 2018.

Yu-Chee Tseng, Sze-Yao Ni, Yuh-Shyan Chen, and Jang-Ping Sheu. The broadcast storm problem in a mobile ad hoc network. *Wireless networks*, 8(2-3): 153–167, 2002.

John Tsiligaridis and Cesar Flores. Reducing energy consumption for distributed em-based clustering in wireless sensor networks. *Procedia Computer Science*, 83:313–320, 2016.

Amos Tversky. Features of similarity. *Psychological review*, 84(4):327, 1977.

Scott White and Padhraic Smyth. A spectral clustering approach to finding communities in graphs. In *Proceedings of the 2005 SIAM international conference on data mining*, pages 274–285. SIAM, 2005.

Christopher R Wren, David C Minnen, and Srinivasa G Rao. Similarity-based analysis for large networks of ultra-low resolution sensors. *Pattern Recognition*, 39(10):1918–1931, 2006.

Lu Xin and Mu Zhu. Stochastic stepwise ensembles for variable selection. *Journal of Computational and Graphical Statistics*, 21(2):275–294, 2012.

Dong Xu, Tat-Jen Cham, Shuicheng Yan, and Shih-Fu Chang. Near duplicate image identification with patially aligned pyramid matching. In *2008 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–7. IEEE, 2008.

Dongkuan Xu and Yingjie Tian. A comprehensive survey of clustering algorithms. *Annals of Data Science*, 2(2):165–193, 2015.

Shibai Yin, Yiming Qian, and Minglun Gong. Unsupervised hierarchical image segmentation through fuzzy entropy maximization. *Pattern Recognition*, 68: 245–259, 2017.

FW Young and RM Hamer. Theory and applications of multidimensional scaling. *Hillsdale, NJ: Eribaum Associates*, 1994.

Tianqi Yu, Xianbin Wang, and Abdallah Shami. Recursive principal component analysis-based data outlier detection and sensor data aggregation in iot systems. *IEEE Internet of Things Journal*, 4(6):2207–2216, 2017.

Liu Yueyang, Ji Hong, and Yue Guangxin. An energy-efficient pegasis-based enhanced algorithm in wireless sensor networks. *China Communications*, pages 91–97, 2006.

Li Zeng, Ling Li, Lian Duan, Kevin Lu, Zhongzhi Shi, Maoguang Wang, Wenjuan Wu, and Ping Luo. Distributed data mining: a survey. *Information Technology and Management*, 13(4):403–409, 2012.

Huazhong Zhang, Peipei Chen, and Shulan Gong. Weighted spanning tree clustering routing algorithm based on leach. In *2010 2nd International Conference on Future Computer and Communication*, volume 2, pages V2–223. IEEE, 2010.

Jing Zhou, Aihuang Guo, Branko Celler, and Steven Su. Fault detection and identification spanning multiple processes by integrating pca with neural network. *Applied Soft Computing*, 14:4–11, 2014.

Roie Zivan and Amnon Meisels. Message delay and discsp search algorithms. *Annals of Mathematics and Artificial Intelligence*, 46(4):415–439, 2006.